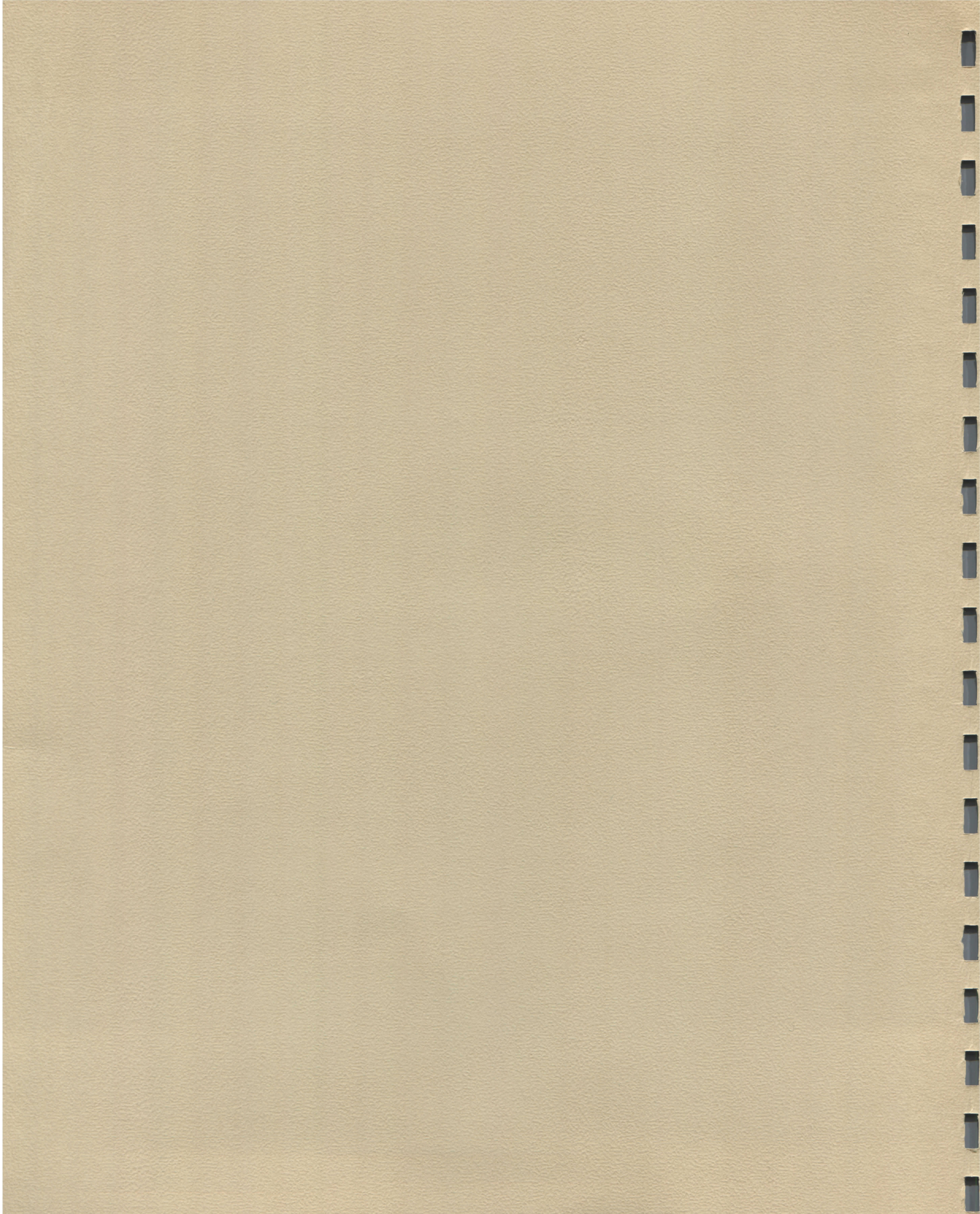


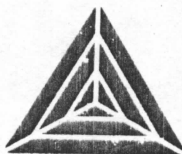
THE MISOSYS MARK IV  
COLLECTION







# The Mark \* Collection



This is a limited edition two-disk collection of software with documentation which bundles together twenty products previously sold individually by MISOSYS, Inc., and Logical Systems Inc. A collection is sold as-is; with no warranties implied. All Mark III programs will work with LDOS 5.1.4; no utilization of LDOS 5.3 enhancements are guaranteed nor will any of the programs be modified in any way by MISOSYS, Inc. All MARK IV programs will work with TRSDOS 6.2 ; no utilization of LS-DOS 6.3 enhancements are guaranteed nor will any of the programs be modified in any way by MISOSYS, Inc. This is a final sale; no returns are accepted.

You get the following programs and files in a Mark III collection:

ALLOC/CMD, BINCONV/CMD, BINPLAY/CMD, BINPRINT/CMD, BSORT51/CMD, BSORT53/JCL, CALC/ASM, CALC/FLT, CMDFILE/CMD, COMMI/ASM, COMMI/FLT, CONVCPM/CMD, CRLF/FLT, CTLG/FLT, CVTEXT/CMD, DD/CMD, DED/CMD, DESCRIBE/CMD, DICTATE/ASM, DICTATE/FLT, DMP2005A/FIX, DOAUTO/CMD, DOCONFIG/CMD, DOEDIT/FLT, DOSAVE/FLT, DOSPEED/ASM, DOSPEED/FLT, DVORAK/XLT, EBCDIC/XLT, EPBINCAT/CMD, EPBINCAT/FIX, FEDII/CMD, FM/CMD, FTS5/CMD, FTS5/HLP, HELPGEN/CMD, IFC/CMD, IFCLIST/CMD, IOMON/CMD, KISTORE/FLT, KSMPLUS/ASM, KSMPLUS/FLT, LCOUNT/ASM, LCOUNT/FLT, LINEFEED/ASM, LINEFEED/FLT, LISTBAS/ASM, LISTBAS/FLT, LOWER/ASM, LOWER/FLT, MARGIN/ASM, MARGIN/FLT, MAXLATE/ASM, MAXLATE/FLT, MEMDIR/CMD, MEMDISK/DCT, MONITOR/ASM, MONITOR/FLT, NAME/CMD, NODAM/CMD, PAGEPAWS/ASM, PAGEPAWS/FLT, PARMDIR/CMD, PDS/CMD, PRTOGGLE/CMD, PTRACE/CMD, RD40/CMD, REMOVE/CMD, RSBINCAT/CMD, SLASH0/ASM, SLASH0/FLT, SLOSTEP/ASM, SLOSTEP/DCT, STRACE/CMD, STRIP7/ASM, STRIP7/FLT, STRIPCNT/ASM, STRIPCNT/FLT, TITLE/ASM, TITLE/FLT, TRAP/ASM, TRAP/FLT, UNKILL/CMD, UPPER/ASM, UPPER/FLT, VIDSAV/CMD, WC/CMD, XLATE/ASM, XLATE/FLT, XONXOFF/FLT, ZCAT/CMD, ZGRAPH/CMD, ZSHELL/CMD

You get the following programs and files in a Mark IV collection:

ALLOC/CMD, ALTDISK/CMD, ALTLD/CMD, ALTRES/CMD, BE/LMF, BINCONV/CMD, BINPLAY/CMD, BINPRINT/CMD, BSORT/CMD, CALC/CMD, CRLF/FLT, CTLG/FLT, CVT324/CMD, CVTEXT/CMD, DD/CMD, DED/CMD, DESCRIBE/CMD, DMP2006A/FIX, DOCONFIG/CMD, DOEDIT/FLT, DOSAVE/FLT, EPBINCAT/CMD, FKEY/CMD, FM/CMD, FTS/CMD, FTS/HLP, HANDY/CMD, HELPGEN/CMD, IFC/CMD, IFCLIST/CMD, INSTALLB/CMD, IOMON/CMD, KISTORE/FLT, LSCOMP/CMD, LSFEDII/CMD, LSQFB/CMD, MAPPER/CMD, MEMDIR/CMD, MINIDOS/FLT, MOD324/CMD, NAME/CMD, OD/CMD, PARMDIR/CMD, PDS/CMD, PROCESS/CMD, PROCURE/CMD, PRTOGGLE/CMD, PTRACE/CMD, RD40/CMD, RSBINCAT/CMD, STRACE/CMD, SWAP/CMD, UNREMOVE/CMD, WC/CMD, XONXOFF/FLT, ZCAT/CMD, ZGRAPH/CMD, ZSHELL/CMD

The complete Volume II six-issue set of the LDOS QUARTERLIES is also included in each collection.

## MISOSYS, Inc.







## BEEP - BASIC Enhancement and Extension Package

The BEEP package will add enhancements to the BASIC supplied with TRSDOS 06.02.00 for the Model 4/4P. It is designed and tested to work only with BASIC version 01.01.00, running under TRSDOS 06.02.00.

The diskette supplied with this package contains two files: BE/LMF and INSTALLB/CMD. The enhancements need to be installed on a copy of BASIC. To perform the installation, use the INSTALLB/CMD program. INSTALLB will append the enhancements (which are contained in the BE/LMF file) onto the end of the BASIC/CMD file.

### Installing BEEP

Simply enter this command at the TRSDOS Ready prompt to install the enhancements:

INSTALLB

A prompt will appear, requesting the drive number which contains the program file BASIC/CMD. After answering this prompt accordingly, the installation will take place on the specified drive. If the installation cannot be done, an informative message will be displayed, and the installation procedure will abort. Some of the situations which may cause the installation procedure to abort are:

- 1) The drive specified does not contain BASIC.
- 2) The drive specified is write protected.
- 3) The BASIC contained on the drive is not version 01.01.00.
- 4) The BASIC contained on the drive has the "enhancements" installed.

### Enhancements

BEEP adds several enhancements to BASIC version 01.01.00. Each of the following BASIC commands may be represented as single characters. When using a single character command, the effect will be much the same as when the entire word is used. This abbreviated form is only acceptable when typed on a command line, and cannot be incorporated in a BASIC program line or JCL file. The abbreviations are:

- A - represents the AUTO command
- D - represents the DELETE command
- E - represents the EDIT command
- L - represents the LIST command

The following are some examples of using abbreviated commands:

- E20 - Edit line number 20
- L.-50 - List all program lines starting with the current program line to line 50, inclusive.
- A15,3 - Enter the AUTO line entry mode, starting at line 15, with a line increment of 3.
- D5-50 - Delete lines 5 through 50, provided that line numbers 5 and 50 exist.

Notes: Use of a space is not required when entering abbreviated commands. Due to the manner in which the BASIC command interpreter works, the period cannot be used for the last line in a line number range (e.g. the command L5-. will produce an error - however, the command L.-50 is acceptable).



The following commands are implemented by pressing the indicated key as the first character in the command line. No carriage return is necessary; the indicated action will take place immediately. Note that any of the following single key commands must be the first character entered on the command line.

- <.> (period) - Performs the same function as LIST.<ENTER>, which will cause the "current" line to be listed.
- <,> (comma) - Performs the same function as EDIT.<ENTER>, which will activate the "edit mode" for the "current" line.
- <UP ARROW> - List the next lower numbered line in the program. The line listed will become the new "current" line.
- <DOWN ARROW> - List the next higher numbered line in the program. The line listed will become the new "current" line.
- <LEFT ARROW> - List the first line of the program and set the "current" line to the first line.
- <DOWN ARROW> - List the last line of the program, and set the "current" line to the last line.

Two new commands are included with the BEEP enhancements. The <C> command will allow the duplication (copying) of a specified existing line number to a non-existing line number. The <M> command will allow an existing line number to be moved to a non-existing number. As with the single letter abbreviations, these commands are only valid on the command line. The syntax for using the <C> and <M> commands is:

Caaaa,bbbb  
Maaaa,bbbb

In each case, aaaa represents the existing line number and bbbb represents the new line number which will be created. The line number specified by aaaa must exist, and the line number specified by bbbb cannot exist. If either of these conditions is not met, a Syntax Error will be generated, and no line movement or changes will occur. A comma <,> must be used as a separator between the two line numbers. The following examples will illustrate the use of each command.

Assume that you currently have a line number 20 in your program, and you wish to make a duplicate copy of this line as line number 35 (where line 35 does not exist), so that line 20 and line 35 are identical. To perform this line duplication, use this command:

C20,35

Using these same assumptions (i.e. line number 20 exists and line number 35 does not exist), suppose that you wish to move line number 20 to line number 35 (in essence, change the number of the line from 20 to 35). To perform this line movement, use this command:

M20,35

Notes: When either moving or copying lines, line number 0 is invalid (for either aaaa or bbbb), and will cause an error if used. Also, moving a line will NOT change any internal program references to the old line number. In the above example, any lines containing internal references to line 20 would have to be manually edited to reflect the new line number.

The last enhancement found in BEEP deals with loading and saving program files. BEEP contains enhancements to perform high-speed loading/saving of programs. There will be a significant decrease in the amount of time it takes to "SAVE" or "LOAD" a BASIC program file, provided that the load or save is done in compressed format (i.e. for loading, the program was not saved in ASCII; for saving, the "A" parameter was not specified).



**BSORT**

BSORT is a high speed sort utility for sorting BASIC arrays. Any type of array (integer, single/double precision or string) may be sorted. Sorts can be performed on one and two dimensional arrays. The following syntax is used to initiate a sort. Note: "Integer Numbers" refers to integer variables or constants.

```
=====
SYSTEM"RUN BSORT NUM%,*IND%,PSA(x),parm,parm,...,parm"
SYSTEM"RUN BSORT $STRVAR$"

NUM%      Number of elements to sort (an integer number).

*IND%(x)  Optional single dimension integer array. If not
          used, re-ordering of elements will occur in the
          array being sorted. If used, the sort will
          generate an index array containing element
          numbers of the sorted array, and no re-ordering
          of "sorted arrays" will occur.

PSA(x)    Primary sort array. An optional <+> or <-> may
          precede the array name to indicate the direction
          (ascending or descending order) of the sort. If
          not specified, <+> is assumed. A declaration tag
          (!, #, $, %) must be used for any array specified.
          A subscript must be specified, representing the
          first element number to be sorted. It must be an
          integer number.

Optional parameters are as follows:

SSA(x)    Secondary sort array. If used, a <+> or <-> must
          precede the array name. The sort key used will
          include corresponding information from the
          primary and secondary arrays. Any re-ordering of
          the primary array will cause a corresponding
          re-ordering of the secondary array. More than
          one may be used. A subscript is required if the
          secondary array is two dimensional.

TA        Tag array. Any re-ordering in the primary array
          will cause a corresponding re-ordering in a tag
          array. A tag array cannot be preceded by a <+>
          or <->, and may only appear after all secondary
          array definitions. More than one may be used.

(s,n)     Mid-string information. Valid only with STRING
          arrays. If specified, it must immediately follow
          the array information, and cannot be used with
          tag arrays. If specified, the sort key will
          begin at position s in the string, for n
          characters, where s and n are integer numbers.

$STRVAR$  Optional non-array string variable containing
          the sort parameters. Must be used if the length
          of the sort command (i.e. the number of chars.
          within the quote marks) exceeds 79.
=====
```



BSORT can be used to perform many different sorting tasks from within a BASIC program. Only variables and arrays that have been "established" can be used; BSORT cannot allocate memory for variables or arrays. If an un-dimensioned array is used in a sort command, an error will be generated. The following examples illustrate the many different types of sorts which can be performed.

### Sorting a Single Dimension Array

Sorting a single dimension array is the simplest type of sort. An integer, single/double precision, or string array may be sorted. In order to sort a one dimension array, two parameters must be passed to BSORT: the number of elements to be sorted, and the starting position in the primary sort array. As an example, assume that the following string array exists in memory:

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)	A\$(6)
SMITH	JONES	BROWN	WILLIAMS	JOHNSON	GREEN

To sort the array, this command would be used, with the results shown below.

SYSTEM"RUN BSORT 6,+A\$(1)"

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)	A\$(6)
BROWN	GREEN	JOHNSON	JONES	SMITH	WILLIAMS

In the example, the number of elements to be sorted was specified as six, with the sort being performed on array A\$ (the primary sort array), starting at element 1.

In this type of sort, a re-ordering of elements takes place. The sequence is in ascending order, so that the value of A\$(1)<A\$(2)<A\$(3) etc. The plus sign <+> in front of the primary sort array indicates that the direction of the sort is to be in ascending order. In this case, the plus sign is optional; if the primary sort array appears without a sign preceding it, the sort will be in ascending order.

Sorting may also be performed so that the re-ordering is in descending order. This is accomplished in the sort command by using a minus sign <-> in front of the primary sort array. Thus, after executing the sort command:

SYSTEM"RUN BSORT 6,-A\$(1)"

the value of A\$(1) would be "WILLIAMS"; the value of A\$(6) would be "BROWN".

Note that in the above examples, the number of elements to be sorted (6) and the starting array position (1) were specified as integer constants. Any integer constant which needs to be passed to BSORT can be specified as a simple (non-array) variable. The only restriction in using a variable as a value passed to the sort utility is that it must be an integer type, with the type declaration tag (%) explicitly present. DEF statements (e.g. DEF INT) may be used; but the variable as used in the sort command must have a type declaration tag.

Arrays used in BSORT must have a type declaration tag. In the above examples, an error would occur if the commands DEF STR A: DIM A(6) were issued and the following sort invoked, since the <\$> declaration tag was not present.

SYSTEM"RUN BSORT 6,A(1)"

When sorting an array, any part of the array may be sorted for any number of elements. Assuming from the above examples that A\$ is dimensioned to have 7 elements {A\$(0) through A\$(6)}, the following sort can be executed.

```
NM%=4:PO%=2
SYSTEM"RUN BSORT NM%,A$(PO%)"
```

This sequence of commands would cause elements 2 through 5 to be sorted in ascending order, and would leave elements 0,1 and 6 untouched.

An error will be generated if sorting is forced beyond the dimensioned length of the array. In the above example, if PO% is 2, an error will be generated if NM% is assigned a value greater than 5.

### Using Secondary Sort Arrays

More than one array may be used to determine the results of a sort operation. Secondary sort arrays can be specified after the primary sort array, and will be included in the sort (i.e. they will aid in determining the direction of the sort and will be re-ordered in conjunction with the primary sort array).

For example, assume that the following arrays are currently active in memory.

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)	A\$(6)
SMITH	JONES	JONES	WILLIAMS	JOHNSON	JONES

F\$(1)	F\$(2)	F\$(3)	F\$(4)	F\$(5)	F\$(6)
SAMMY	BILLY	BETTY	RICHARD	CHARLES	BOBBY

The array A\$ represents a list of last names, while the array F\$ contains the corresponding first name. It is desired to create a sorted list of these names in ascending order, where the first name is used to determine the sorted order when the last names are the same. The following sort command may be used to accomplish this task, with the results shown below.

```
SYSTEM"RUN BSORT 6,A$(1),+F$"
```

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)	A\$(6)
JOHNSON	JONES	JONES	JONES	SMITH	WILLIAMS

F\$(1)	F\$(2)	F\$(3)	F\$(4)	F\$(5)	F\$(6)
CHARLES	BETTY	BILLY	BOBBY	SAMMY	RICHARD

The array F\$ is a secondary sort array. It is used in the sorting process to determine the sorted order when a direct match is found in the primary array. When a secondary sort array is specified, a direct correlation between elements in the primary array is assumed. Any re-ordering which occurs in the primary array will also occur in the secondary array. Thus in this example, the first names were carried along in the sort with the last names, and any exact matches on the last names caused the first names to be sorted.



There are some points that need to be made with respect to syntax and usage of secondary sort arrays. When dealing with a single dimension secondary array, it must be separated from the primary array with a comma. Additionally, no subscript number is required. Any re-ordering which occurs will be done according to the element number in the primary array (i.e. element 1 in the primary array corresponds to element 1 in the secondary array). Furthermore, secondary arrays must be dimensioned as high as the largest sorted element number in the primary array. For example, if a primary array is dimensioned to have 50 elements (0-49) and a secondary array is dimensioned to have 10 elements (0-9), a sort using both arrays could only be performed up to and including element nine. An error will be generated if the sort should go beyond the highest allowable element number of either the primary or secondary array.

Unlike primary arrays, use of a direction sign (+ or -) is mandatory when specifying a secondary array. The direction of the sort in a secondary array does not have to match that of the primary array. Using the above arrays (A\$ and F\$), the following sort command would produce the results shown below.

SYSTEM"RUN BSORT 6,+A\$(1),-F\$"

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)	A\$(6)
=====					
JOHNSON	JONES	JONES	JONES	SMITH	WILLIAMS
=====					
F\$(1)	F\$(2)	F\$(3)	F\$(4)	F\$(5)	F\$(6)
=====					
CHARLES	BOBBY	BILLY	BETTY	SAMMY	RICHARD
=====					

Note that the directioning of the secondary sort array (in descending order) did not affect the re-ordering of the primary array (ascending order). However, any exact matches in the primary array caused the secondary array (first name array) to be sorted in descending order.

### Using Multiple Secondary Arrays

The concept of using more than one secondary array does not differ greatly from using just one secondary array. In terms of syntax, commas must separate subsequent secondary arrays. The same restrictions also apply when more than one secondary array is used (i.e. the mandatory direction sign and the dimensioned lengths of the arrays).

The important point to note is that the order in which the arrays are entered on the sort command line may have an effect on the results of the sort. That is, the first secondary array specified will be the first array used to determine the results of the sort. As an example, examine the three arrays that are shown on the next page.



```

A$(1)  A$(2)  A$(3)  A$(4)  A$(5)  A$(6)  A$(7)
=====
| SMITH | BROWN | JONES | JONES | JONES | JONES | JONES |
=====

F$(1)  F$(2)  F$(3)  F$(4)  F$(5)  F$(6)  F$(7)
=====
| SAMMY | ROBBY | JOHN  | JAKE  | JOHN  | HERB  | HERM  |
=====

I$(1)  I$(2)  I$(3)  I$(4)  I$(5)  I$(6)  I$(7)
=====
| 1001  | 1002  | 1003  | 1004  | 1005  | 1006  | 1007  |
=====

```

Array A\$ contains last names, Array F\$ contains first names and Array I\$ contains ID numbers. Consider the results of the following sort command (shown below).

SYSTEM"RUN BSORT 7,A\$(1),+F\$,-I\$"

```

A$(1)  A$(2)  A$(3)  A$(4)  A$(5)  A$(6)  A$(7)
=====
| BROWN | JONES | JONES | JONES | JONES | JONES | SMITH |
=====

F$(1)  F$(2)  F$(3)  F$(4)  F$(5)  F$(6)  F$(7)
=====
| ROBBY | HERB  | HERM  | JAKE  | JOHN  | JOHN  | SAMMY |
=====

I$(1)  I$(2)  I$(3)  I$(4)  I$(5)  I$(6)  I$(7)
=====
| 1002  | 1006  | 1007  | 1004  | 1005  | 1003  | 1001  |
=====

```

The primary sort occurred on the last name (ascending order). If an exact match occurred in the last name, the first name was used to determine the order (ascending order). If two people had identical first and last names, the ID number was then used, and sorted in descending order.

If the secondary arrays were "switched" on the command line, the results obtained would be quite different. Observe this sort command and associated results.

SYSTEM"RUN BSORT 7,A\$(1),-I\$,+F\$"

```

A$(1)  A$(2)  A$(3)  A$(4)  A$(5)  A$(6)  A$(7)
=====
| BROWN | JONES | JONES | JONES | JONES | JONES | SMITH |
=====

F$(1)  F$(2)  F$(3)  F$(4)  F$(5)  F$(6)  F$(7)
=====
| ROBBY | HERM  | HERB  | JOHN  | JAKE  | JOHN  | SAMMY |
=====

I$(1)  I$(2)  I$(3)  I$(4)  I$(5)  I$(6)  I$(7)
=====
| 1002  | 1007  | 1006  | 1005  | 1004  | 1003  | 1001  |
=====

```



Note that the last names did sort in ascending order. However, since the I% array appears immediately after the primary array, any identical match found on the last name caused the next sort criteria to be taken from the I% array, with the results being determined in descending order.

### Using Tag Arrays

In addition to using secondary arrays, Tag Arrays may be specified on a sort command line. They are similar to Secondary sort arrays, with the exception that the information contained in them has no bearing on the results of the sort. If a tag array is used, any re-ordering which occurs in the primary sort array will also occur in a tag array.

Tag arrays are distinguished from secondary arrays by the lack of a direction sign. If an array (other than the primary array) has no direction sign, it is taken to be a tag array. If both tag and secondary arrays are to be used, ALL secondary arrays must be defined on the sort command line prior to any tag array definitions. Subsequent tag arrays must be separated by commas, and no subscript number is required.

Consider a last name - first name example, where the following arrays have been defined in memory.

```

A$(1)  A$(2)  A$(3)  A$(4)  A$(5)  A$(6)
=====
| JONES | JONES | JONES | WILLIAMS | JOHNSON | JONES |
=====

F$(1)  F$(2)  F$(3)  F$(4)  F$(5)  F$(6)
=====
| ROBIN | BILLY | BETTY | RICHARD | CHARLES | BOBBY |
=====

```

A typical sort command which makes use of F\$ as a tag array could be represented by the following, with the results shown below.

```

SYSTEM"RUN BSORT 6,A$(1),F$"

A$(1)  A$(2)  A$(3)  A$(4)  A$(5)  A$(6)
=====
| JOHNSON | JONES | JONES | JONES | JONES | WILLIAMS |
=====

F$(1)  F$(2)  F$(3)  F$(4)  F$(5)  F$(6)
=====
| CHARLES | ROBIN | BILLY | BOBBY | BETTY | RICHARD |
=====

```

Note that the last names are sorted in correct ascending order. However, since F\$ was used as a tag array, it did not affect the results of the sort, and only a re-ordering of the data occurred. The order of the items shown in the F\$ array is related to the re-ordering of the A\$ array. Whenever exact data matches occur (as is the case with the name JONES), re-ordering is done in a random fashion. If additional sort information is required, it should be specified as a secondary sort array. If information is only to be "moved along with" sorted information, it may be specified as a tag array.



## MID\$ Sorting

When sorting string arrays, an optional MID\$ (mid string) parameter may be specified with primary and/or secondary arrays. This will allow sorting to be done on a string array using only a specified part of the string. If re-ordering is performed, the entire string element will be "moved".

As an example, consider the following two string arrays.

L\$(0)	L\$(1)	L\$(2)	L\$(3)	L\$(4)
=====				
D. BROWN	R. SMITH	T. JONES	R. SMITH	P. JONES
=====				

F\$(0)	F\$(1)	F\$(2)	F\$(3)	F\$(4)
=====				
BR, DALE	SM, ROB	JO, TERRY	SM, RICH	JO, PETE
=====				

Array L\$ contains a first name initial, followed by a period, a space and a last name. Array F\$ contains the first two characters of the last name, followed by a comma, a space, and the first name. With a sort to be done in the order of last name - first name, this sort command could be used, with the results shown below.

```
SYSTEM"RUN BSORT 5,L$(0)(4,7),+F$(5,6)
```

L\$(0)	L\$(1)	L\$(2)	L\$(3)	L\$(4)
=====				
D. BROWN	P. JONES	T. JONES	R. SMITH	R. SMITH
=====				

F\$(0)	F\$(1)	F\$(2)	F\$(3)	F\$(4)
=====				
BR, DALE	JO, PETE	JO, TERRY	SM, RICH	SM, ROB
=====				

In this sort command, the primary sort array is the L\$ array, while F\$ is a secondary array. Both arrays are sorted in ascending order.

The MID\$ information immediately follows the subscript position for the primary array. It is enclosed within parentheses, and consists of two integer numbers. The first number specifies the position in the string where the sort criteria begins. Here, the strings in the L\$ array are to be sorted starting with the fourth character (i.e. the first character of the last name). The second number tells the sort utility the number of characters to include in the sort from the starting position. Here, seven characters of each string (starting at position four of the string) will comprise the sort key for the primary sort array.

Similarly, MID\$ information has been supplied with the secondary sort array. Using the F\$ array, the sort key will begin at position 5 (i.e. the first character of the first name) in each element of the array, and extend for 6 characters into each string. Thus, the two arrays are sorted in the order of last name - first name.

Several points need to be made with respect to MID\$ sort information. It must always immediately follow the last piece of information associated with the array (i.e. no comma separator is used). When sorting single dimension arrays, this will come after the closing parenthesis of the subscript number for the primary array, and after the declaration tag of the secondary array.

BSORT will NOT check to see if the MID\$ values are valid for any string, with the exception that they must not exceed 255. If the starting MID\$ position exceeds the entire length of the string in question, a "null" value will be used for that particular element of the array. If the starting MID\$ position is within the string, but the number of characters to use for sort criteria is more than what is remaining in the string, only the remaining characters will be used.

For example, A\$(1)="HI", A\$(2)="BYE" and A\$(3)="THIS IS THE END". The following sort commands will produce the results shown below.

Sort Command	Ordering of Elements_
1. SYSTEM"RUN BSORT 3,A\$(1)(1,3)"	2,1,3
2. SYSTEM"RUN BSORT 3,A\$(1)(2,4)"	3,1,2
3. SYSTEM"RUN BSORT 3,A\$(1)(3,2)"	1,2,3

In example 1, the first 3 characters of each string are used in the sort to determine the results. In example 2, the second through fifth characters of each string are used. In example 3, characters three and four of each string are used. Since the first array position only has a length of two characters, its sort value is "null", and so it was sorted "first" (in ascending order).

### Generating an Index Array

The previous examples illustrated methods by which arrays were sorted into either ascending or descending order. With those sorts, the array data was re-ordered, so that physical access of the array (by ascending/descending element numbers) was required to see the sorted results. In some cases (such as reading data into an array from a random access file) it may not be desirable to actually re-order an array when "sorting". Thus, BSORT may also be used to generate index arrays. BSORT will initialize the index array to contain the element numbers of the array to sort. The sort will re-order the index array, so that the values in the index array will form a list of pointers to the "sorted" elements of the primary array. For example, assume that the following arrays are currently in memory:

P\$(1)	P\$(2)	P\$(3)	P\$(4)	P\$(5)	P\$(6)	P\$(7)
=====	=====	=====	=====	=====	=====	=====
WILLIAMS	SMITH	JONES	BROWN	GREEN	JOHNSON	RICH
=====	=====	=====	=====	=====	=====	=====

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
=====	=====	=====	=====	=====	=====	=====
0	0	0	0	0	0	0
=====	=====	=====	=====	=====	=====	=====

The sort command listed below could be used to create the index array I%.

SYSTEM"RUN BSORT 7,\*I%(1),P\$(1)"

P\$(1)	P\$(2)	P\$(3)	P\$(4)	P\$(5)	P\$(6)	P\$(7)
=====	=====	=====	=====	=====	=====	=====
WILLIAMS	SMITH	JONES	BROWN	GREEN	JOHNSON	RICH
=====	=====	=====	=====	=====	=====	=====

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
=====	=====	=====	=====	=====	=====	=====
4	5	6	3	7	2	1
=====	=====	=====	=====	=====	=====	=====



Notice that the sort command did not alter the primary sort array (P\$). However, the values in the index array (I%) changed to reflect proper access order of the primary array. Since I%(1) has a value of 4, the fourth element of the P\$ array is the first element to access if ascending sorted order is desired. It is now simple to print the P\$ array contents in sorted order, using I% as an index.

```

FOR L%=1 TO 7
PRINT P$(I%(L%))      or      M%=I%(L%):PRINT P$(M%)
NEXT L%                NEXT L%

```

When using an index sort, the index array must immediately follow the number of items to sort on the command line, and must be preceded by an asterisk <\*> which indicates that an index array is to be generated. The index array MUST be a one dimension integer type with the declaration tag used explicitly. A subscript number must be used with the index array; it will indicate the starting position of the indexed information. This subscript may be either an integer constant or a simple integer variable. Finally, the index array must be dimensioned large enough to contain all index values generated - the number of items sorted. Failure to adhere to these guidelines will more than likely generate an error.

Regarding the subscript number used with the index array, in most cases it will be parallel to the subscript number specified in the sorted array. However, it is not mandatory that these two subscript numbers be the same. As an example, assume that the following integer array exists in memory.

```

I%(1) I%(2) I%(3) I%(4) I%(5) I%(6) I%(7) I%(8) I%(9) I%(10)
=====
| 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
=====

```

Using this as an index array, the following sort command on the P\$ array (see previous example) will produce the results shown.

```
SYSTEM"RUN BSORT 4,*I%(6),P$(2)"
```

```

I%(1) I%(2) I%(3) I%(4) I%(5) I%(6) I%(7) I%(8) I%(9) I%(10)
=====
| 100 | 200 | 300 | 400 | 500 | 4 | 5 | 3 | 2 | 1000 |
=====

```

This will sort four elements of the P\$ array (elements 2 through 5), and store the index information in the I% array, starting at element 6. Note that elements 1-5 and 10 in the index array were unaffected by the sort. The numbers stored in the index array correspond to the element numbers that were sorted (2 through 5).

If the I% array was initially dimensioned to contain 11 elements (0-10), the following sort command would cause an error:

```
SYSTEM"RUN BSORT 4,*I%(8),P$(2)"
```

The error would be caused by trying to store index information beyond the end of the index array (i.e. Element #11 of the I% array does not exist).

Indexed sorts may be performed using all of the previously defined sort parameters (i.e. MID\$ and secondary arrays). The syntax for such sorting would remain the same, with the exception of the index array being specified in the sort command. No re-ordering will be done on any array used in an indexed sort. Thus, it would be meaningless to use tag arrays in an indexed sort; although no error would be generated, a tag array will not be affected by an indexed sort.

## Sorting Two-dimensional Arrays

BSORT supports the use of two dimensional arrays as any type of array (primary, secondary, tag) in a sort command. This section will discuss several variations of using two dimensional arrays.

Throughout the documentation, examples have been given of various sort procedures using single dimension arrays. The array illustrations have always been depicted in a "horizontal" fashion, representing one row of information with multiple columns. Sorting a one dimension array implies that a row of the array (in this case the only row of the array) be used as the key information, with individual columns being re-ordered (or indexed) to satisfy the requirements of the sort.

This same concept can be carried over to two dimensional arrays. An individual row of the array is specified, from which the key (sort) information is retrieved. Additionally, a starting column number is specified, and the number of elements to be sorted represents the number of columns involved in the sort. If re-ordering is required, an entire column of data is "moved".

As an example, assume that this array (A\$) has been established in memory.

		COLUMN				
		1	2	3	4	5
R	1	DALE	DAN	DON	DICK	DOCK
	2	BROWN	JONES	SMITH	GREEN	PETERS
	3	25	34	19	53	42
O	4	BOSTON	BUTTE	BALT	PHIL	PITT
	5	03021	78654	23376	19769	16511
W	6	MA	MT	MD	PA	PA
	7	REP	REP	CLIENT	ADV	STOCK

If it was desired to sort this array by last name in ascending order, the following sort command could be entered, with the results shown below.

SYSTEM"RUN BSORT 5,A\$(2,1)"

		COLUMN				
		1	2	3	4	5
R	1	DALE	DICK	DAN	DOCK	DON
	2	BROWN	GREEN	JONES	PETERS	SMITH
	3	25	53	34	42	19
O	4	BOSTON	PHIL	BUTTE	PITT	BALT
	5	03021	19769	78654	16511	23376
W	6	MA	PA	MT	PA	MD
	7	REP	ADV	REP	STOCK	CLIENT

Several points can be drawn from this example. The total number of items to sort is 5. Row 2 is designated as containing the information to sort. The sort will begin at column 1 (in row 2) and continue for a total of 5 columns. If a re-ordering is to take place, all information in the given column is "moved" (essentially, the two columns involved in the re-ordering are "swapped").

If the A\$ array were used as it appeared initially (see Example 1), and the following sort command was issued:

SYSTEM"RUN BSORT 2,A\$(5,4)"



A swap of columns 4 and 5 would be performed. This sort would use information in row 5 as the key. The sort would begin at column 4, and include 2 columns (i.e. columns 4 and 5). Since 16511 is less than 19769, a re-ordering would occur.

Assume once more that the A\$ array exists in memory as shown in Example 1. It is desired to generate an index array (I%), where the information in row 3 is sorted in descending order. The following sort command would accomplish this, with the results shown below.

SYSTEM"RUN BSORT 5,\*I%(1),-A\$(3,1)"

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)
4	5	2	1	3

Note that when indexing a two dimensional array, the column position of the sorted array is stored in the index array. The sorted array remains unchanged.

### Using Two Dimensional Secondary and Tag Arrays

The concept behind sorting two dimensional arrays carries over to the use of two dimensional secondary and tag arrays. In both instances, the number of rows is insignificant. The number of columns in either a secondary or tag array must be as large (or greater than) the highest column number to be sorted in the primary array.

In the case of a tag array, no subscript is required. Re-ordering of columns in the tag array will correspond to those re-ordered in the primary array. The entire column will be "moved", regardless of the number of rows in the array.

The same re-ordering rules apply to two dimensional secondary arrays. However, a subscript must be included with the secondary array. The subscript will be the row number from which key information is to be taken.

Let us assume that the following arrays exist in memory.

A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)
BROWN	ADAMS	BROWN	ADAMS	BROWN

ARRAY B\$						
		1	2	3	4	5
R	1	PRES	VP	MGR	SALES	DIST
O	2	25	53	34	42	19
W	3	DALE	DOCK	DAN	DICK	DON

The A\$ array is to be the primary array, and row three of the B\$ array will be used as secondary sort information. The following sort command would yield these results (i.e. primary sort by last name, secondary sort by first name), with the sorted arrays being shown on the next page.

SYSTEM"RUN BSORT 5,A\$(1),+B\$(3)

	A\$(1)	A\$(2)	A\$(3)	A\$(4)	A\$(5)
	=====	=====	=====	=====	=====
	ADAMS	ADAMS	BROWN	BROWN	BROWN
	=====	=====	=====	=====	=====
B\$	-----	-----	-----	-----	-----
	1	2	3	4	5
R	1  SALES	VP	PRES	MGR	DIST
O	2  42	53	25	34	19
W	3  DICK	DOCK	DALE	DAN	DON

Note that any re-ordering which occurred in the primary array forced the entire (corresponding) column in the B\$ array to be moved.

The same re-ordering will occur if a two dimensional tag array is used. A subscript is not required. Entire columns (regardless of the number of rows) will be re-ordered according to the corresponding re-ordering of the primary array.

When using two dimensional secondary arrays, the same array can be used more than once in a sort command, provided that the row specified is different in each case. As a matter of fact, if the primary array is two dimensional, a row different than the primary sort row may be specified as a secondary sort array. In the case of our first example dealing with two dimensional arrays, if it was desired to obtain a sort on this array primarily by last name (row 2) and secondarily by first name (row 1), the following sort command could be used:

```
SYSTEM"RUN BSORT 5,A$(2,1),+A$(1)"
```

This would have the affect of using row 2 of the A\$ array as the primary sort information, while using row 1 of the same array as secondary sort information.

As a final point, it is permissible to use the MID\$ function when dealing with two dimensional secondary arrays. As is the case with a one dimension primary array, the MID\$ information would immediately follow the row subscript of the secondary array. The following example illustrates the syntax that would be used:

```
SYSTEM"RUN BSORT 10,XX%(2,5),+C$(3)(19,8)"
```

Here, XX% is the primary array. The sort will be on row 2 of this array, starting at column 5. It will extend for 10 columns (up to and including column 14). C\$ will be used as the secondary array. Columns 5-14 of row 3 will be used. Within each of these elements, a MID\$ will be performed, so that the string used in the sort will begin at position 19 of each element, and extend for 8 characters.

### Using a Variable to Pass the Sort Command

Depending on the number of parameters specified, the length of an actual sort command may become quite large. A limitation does exist in that the total length of a SYSTEM command cannot exceed 79. For this reason, BSORT allows for the passing of sort parameters in a simple string variable. For example, this command utilizes information contained in PARM\$ as the parameters to use for the sort.

```
PARM$ = "10,*II%(1),-AA$(4,1)(15,20),+AC$(3),-SD$(7)(13,8)"
SYSTEM "RUN BSORT $PARM$"
```

In the command which initiates the sort, a <\$> must precede the string variable containing the sort parameters. In using this type of sort command, the only limitation is that the length of the string cannot exceed 255 characters.



## MOD324

MOD324 is a utility designed to aid in converting programs created under MODEL III BASIC to MODEL 4 BASIC. The MODEL III program must be contained on a diskette formatted by either MODEL 4 TRSDOS/LS-DOS or MODEL III LDOS (use CONV to move the program from a MODEL III TRSDOS diskette to a MODEL 4 TRSDOS/LS-DOS diskette). The following syntax is used (from the DOS Ready level) to perform a conversion:

```
=====
MOD324 filespec1 filespec2 (parm,...,parm)

filespec1  MODEL III program to convert. Must be "Saved"
           in compressed format. If not specified, it
           will be prompted for.

filespec2  File to contain the converted program. If it
           does not exist, the file will be created. If
           it exists, the previous contents of the file
           will be overwritten. If not specified, it
           will be prompted for. When specified on the
           on the command line, it must appear after
           filespec1.

Optional parameters are as follows:

MODIFY      Adjust numeric constants in PRINT@ statements
           to the corresponding value (absolute row and
           column position) on the MODEL 4 video.

CENTER=n    Additional offset value which is added to
           all PRINT@ positions changed by MODIFY. Will
           work only if MODIFY is specified. Will also
           offset numeric constants in PRINT TAB
           statements according to the column position
           of the value entered. Default is 328.
           (4 lines, 8 columns)

PRINT       Send output of possible manual corrections to
           the printer. If not specified, output will be
           to video.

WIDTH=n     Can be used only if PRINT is specified. Will
           determine the maximum number of characters to
           PRINT per line. Default is 80.

abbr:       All parameters may be abbreviated to their
           first character.
=====
```

### I M P O R T A N T   N O T I C E

MOD324 is designed to be used as an aid in converting programs which are currently running on the MODEL III to a format that can be read by the MODEL 4. Some program commands and sequences which function error free on the MODEL III will NOT work on the MODEL 4. Every attempt is made by MOD324 to flag possible error situations that could result. However, there is NO GUARANTEE (implied or otherwise stated) that a program converted by MOD324 will work, even if no "Manual corrections" were indicated.

## Program Description

MOD324 can be used to convert MODEL III programs to a form that can be read by MODEL 4 BASIC. The MODEL III program must be stored in "compressed format" (i.e. it should NOT have been saved in ASCII). MOD324 will create an ASCII file containing many of the necessary changes to allow the program to be run under MODEL 4 BASIC. Some of the conversions that will take place are:.

1. All "Tokenized" key words and symbols found in the MODEL III program will be changed to the corresponding ASCII representation of the key word/symbol in the MODEL 4 file.
2. Spaces will be inserted into the MODEL 4 text where needed. This includes inserting a space after non-function key words (i.e. those that contain no information within parentheses, such as FOR, TO, NEXT), and after variables/constants which precede a key word, and are not separated from the key word by a terminator (e.g. in the sequence IF A%=10 THEN A%=5, a space would be inserted between the <O> of 10 and the <T> of THEN).
3. Any value used in conjunction with a CLEAR statement will be "stripped off". For example, if the statement CLEAR 5000 appeared in the MODEL III program, the resulting statement in the MODEL 4 text would be CLEAR (the function of the CLEAR statement is entirely different on the MODEL 4).
4. Numeric constants used with PRINT@ and PRINT TAB will be adjusted to a corresponding print "position" on the MODEL 4 (if the MODIFY parameter is specified).

There are cases in which no conversions will take place. Any information which appears in the MODEL III program file as ASCII will be left as is. No alterations will be made to either information appearing within quotes, or information following a "Tokenized" REM statement (i.e. the apostrophe character).

Aside from the program conversions that are required, other problems may arise when converting a MODEL III program to run on the MODEL 4. One such source of difficulty is with program statements that exist in MODEL III BASIC but have no meaning on the MODEL 4. Another consideration is in program statements which exist in both BASICs but function differently for one reason or another. Although "translation" of these types of commands would be difficult (if not impossible), MOD324 does provide feedback (i.e. output to the video or printer) on commands that could pose a problem if used with MODEL 4 BASIC.

The following is a list of MODEL III commands that will be "flagged" by MOD324 as possibly needing manual correction.

CLOAD	POINT
CMD	POS
CSAVE	PRINT@
ERR	PRINT TAB
IF (when not followed by THEN)	PRINT #-1 , PRINT #-2
INP	RESET
INPUT #-1 , INPUT #-2	SET
NAME	SYSTEM
OUT	TIME\$
PEEK	USR
POKE	



PRINT statements (in particular PRINT@ and PRINT TAB) receive special consideration when encountered by MOD324. Although these commands are accepted by MODEL 4 BASIC, video output can cause a problem, since the video sizes differ (64x16 vs. 80x24). For this reason, any occurrence of PRINT@ and PRINT TAB statements will be flagged. There are provisions for MOD324 to adjust values associated with these PRINT statements. Refer to the information on the MODIFY and CENTER parameters for further details.

The last situation which will be flagged by MOD324 is when the resulting conversion would cause a program line to exceed the maximum line length. Due to the "expansion" of key words and the insertion of spaces, a MODEL III program line could be converted into a line which is greater than 254 characters (the maximum line length in MODEL 4 BASIC). When this type of situation occurs, the line will be truncated, and any information in the original program line that could not be saved to the MODEL 4 program file would be displayed on the video (or sent to the printer). In this case, a new line will need to be added to the MODEL 4 program, incorporating the "Lost" information. Note: Program logic may be affected by the truncation of a line.

### Program Usage

To perform a program conversion, all that is required is to enter <MOD324> at the DOS Ready level. The following prompts to appear (one at a time).

Input Filespec?  
Output Filespec?

Pressing <BREAK> in response to either prompt will cause a return to DOS Ready. Any error encountered while answering these prompts (e.g. File not in directory or Write protected disk) will cause the appropriate error message to be displayed, after which the same prompt will re-appear. All entries must follow the rules associated with valid filespecs.

The first prompt is for the name of the MODEL III program. Answer this prompt by entering the associated filespec. If a drivespec is not used, a global search of all active drives will be performed. Please note that if the file has an extension, the extension must be specified (i.e. /BAS is NOT assumed).

The second prompt is for the name of the file which will contain the converted program. If the filespec entered does not exist, it will be created. If the filespec does exist, any information previously contained in the file will be overwritten by the converted program text. It is recommended that a drivespec be included with the output filespec, to assure that the file is written to the proper place. If a drivespec is not entered, the output file will be written to either the "first" drive containing the file, or to the first available drive if the file does not exist on any drive in the system.

Both filespecs may be entered on the command line. For example, if the MODEL 4 program TEST/M4 is to be created (on drive 2) from the MODEL III program TEST/BAS (on drive 1), the following command could be entered.

MOD324 TEST/BAS:1 TEST/M4:2

If only one filespec appears on the command line, it will represent the input filespec, and a prompt will appear for the output filespec.

To see the results of performing a conversion, assume that the following program has been created by MODEL III BASIC, and was saved in compressed form using the

filespec SAMPLE/BAS.

```
10 CLEAR5000:DEFINT A-N:DEFSTRS,T
20 CLS:FORL=1TO10
30 PRINTTAB(5)"This is Line";L;"on the MOD III video";TAB(45)"Position
45"
40 NEXT L
```

It is desired to "convert" this program for use on the MODEL 4. The name of the file to contain the converted program is SAMPLE/M4 on drive 2. The following command may be entered to accomplish this.

#### MOD324 SAMPLE/BAS SAMPLE/M4:2

Two results will occur from the above command. An ASCII file containing the converted program will be created, and feedback for possible manual program corrections (if any) will be given. The first consideration is the program file that is created. The following is a listing of the file SAMPLE/M4.

```
10 CLEAR:DEFINT A-N:DEFSTR S,T
20 CLS:FOR L=1 TO 10
30 PRINT TAB(5)"This is Line";L;"on the MOD III video";TAB(45)"Position
45"
40 NEXT L
```

One point to draw from this listing is the insertion of spaces. Spaces will be inserted as needed. This is clearly illustrated in Lines 10, 20 and 30. Note that in Line 40 no space was added, since one already existed (between the <T> of NEXT and the variable L).

Of additional interest is the resulting CLEAR statement in Line 10. Since the value associated with a MODEL 4 CLEAR statement does not dictate the amount of string space to allocate, any value following a CLEAR statement will be stripped.

In terms of the feedback given (of possible manual corrections), the following information would appear on the video as a result of the conversion performed.

The following lines may need manual correction:

```
30 TAB,TAB
```

File output completed

Any "flagged" key word (see the list on Page 2) that appears in the program will be displayed as the output file is being created. The number of the line containing a flagged key word will be displayed, followed by the key words in question. If multiple key words are flagged on a line, they will be separated by commas. In this example, the key words PRINT TAB appeared twice in Line 30. Note that when TAB appears in a manual correction listing, it is taken to be associated with a PRINT TAB sequence. If TAB is used with an LPRINT statement, no flagging will occur.

After MOD324 has created the output file, it is the sole responsibility of the user to make any manual corrections. In this example, the program could be run as is. However, if any key words were flagged that did not exist in MODEL 4 BASIC (such as SET), they would have to be removed. Furthermore, if key words were found that could cause unpredictable results (such as a POKE of video ram), lines containing these statements would also need to be modified.



### PRINT and WIDTH= Parameters

Depending on the length of the program to be converted, the resulting output on manual corrections could become quite lengthy. For this reason, the PRINT parameter has been included. By specifying PRINT, any feedback on possible manual corrections will be sent to the printer (as well as the video).

If PRINT is specified, the WIDTH= parameter may also be used. This will determine the number of characters sent to the printer per line. The default value for WIDTH= is 80. Any value between 9 and 255 may be used.

The printer output will be formatted, so that the line number of a line needing manual corrections will be printed at position 1 (leftmost part) of the line of output. The list of key words will begin at print position 7, and continue for as many key words that exist in the line. If the number of key words to be displayed on the line would cause the WIDTH to be exceeded, the line will be broken at the key word preceding the one causing the "wrap around" (if possible). All remaining key words will then be printed on the next line, starting at print position 7.

Assume it is desired to obtain printed output of possible manual corrections when converting the program SAMPLE/BAS to SAMPLE/M4. The total length of an output line is not to exceed 60 characters. The following command will accomplish this.

MOD324 SAMPLE/BAS SAMPLE/M4 (P,W=60

### MODIFY and CENTER= Parameters

A definite problem can arise with respect to "screen formatting" when converting a MODEL III program to run on the MODEL 4. Consider the following (MODEL III) program, which draws a box on the first 15 lines of the video, prints an informative message on the last line, and blinks a message in the middle of the box.

```
5 CLEAR 2000
10 CLS
20 PRINT@0,CHR$(151);STRING$(62,131);CHR$(171)
30 PRINT@64,CHR$(149):PRINT@127,CHR$(170)
40 PRINT@128,CHR$(149):PRINT@191,CHR$(170)
50 PRINT@192,CHR$(149):PRINT@255,CHR$(170)
60 PRINT@256,CHR$(149):PRINT@319,CHR$(170)
70 PRINT@320,CHR$(149):PRINT@383,CHR$(170)
80 PRINT@384,CHR$(149):PRINT@447,CHR$(170)
90 PRINT@448,CHR$(149):PRINT@511,CHR$(170)
100 PRINT@512,CHR$(149):PRINT@575,CHR$(170)
110 PRINT@576,CHR$(149):PRINT@639,CHR$(170)
120 PRINT@640,CHR$(149):PRINT@703,CHR$(170)
130 PRINT@704,CHR$(149):PRINT@767,CHR$(170)
140 PRINT@768,CHR$(149):PRINT@831,CHR$(170)
150 PRINT@832,CHR$(149):PRINT@895,CHR$(170)
170 PRINT@896,CHR$(181);STRING$(62,176);CHR$(186);
175 PRINT@960,"";TAB(15)"Press Any Key to end this Program";
180 PRINT@473,"Center of Box";
190 I$=INKEY$:IFI$<>"THENEND
200 FORL=1TO30:NEXTL
210 PRINT@473,"";
220 I$=INKEY$:IFI$<>"THENEND
230 FORL=1TO20:NEXTL:GOTO180
```

Assuming that this program has been saved as CENTER/BAS, the following conversion

command will produce the feedback output shown.

MOD324 CENTER/BAS CENTER/M4:3

File CENTER/M4:3

The following lines may need manual correction:

```
20  PRINT@(0)
30  PRINT@(64),PRINT@(127)
40  PRINT@(128),PRINT@(191)
    .
    .
    .
150  PRINT@(832),PRINT@(895)
170  PRINT@(896)
175  PRINT@(960),TAB
180  PRINT@(473)
210  PRINT@(473)
```

In this example, all PRINT@ commands use numeric constants to represent print positions. The converted program (CENTER/M4) could be run without performing manual corrections. However, the results would not produce a box on the screen.

In situations similar to this one, the MODIFY parameter may be used. MODIFY will adjust PRINT@ positions which are represented by numeric constants. The output program file will contain these adjusted values, and the feedback will show both the original and adjusted values. The original PRINT@ position will be divided by 64 to obtain an integer quotient and remainder. These numbers correspond to the row and column of the PRINT@ position, offset from 0. The adjusted PRINT@ value is obtained by multiplying the row value by 80 and adding in the column number.

The following command will perform a conversion of the program CENTER/BAS, incorporating the MODIFY parameter. The feedback output is shown below.

MOD324 CENTER/BAS CENTER/M4:3 (M)

File CENTER/M4:3

The following lines may need manual correction:

```
20  PRINT@(0=>0)
30  PRINT@(64=>80),PRINT@(127=>143)
40  PRINT@(128=>160),PRINT@(191=>223)
50  PRINT@(192=>240),PRINT@(255=>303)
60  PRINT@(256=>320),PRINT@(319=>383)
70  PRINT@(320=>400),PRINT@(383=>463)
80  PRINT@(384=>480),PRINT@(447=>543)
90  PRINT@(448=>560),PRINT@(511=>623)
100  PRINT@(512=>640),PRINT@(575=>703)
110  PRINT@(576=>720),PRINT@(639=>783)
120  PRINT@(640=>800),PRINT@(703=>863)
130  PRINT@(704=>880),PRINT@(767=>943)
140  PRINT@(768=>960),PRINT@(831=>1023)
150  PRINT@(832=>1040),PRINT@(895=>1103)
170  PRINT@(896=>1120)
175  PRINT@(960=>1200),TAB
180  PRINT@(473=>585)
210  PRINT@(473=>585)
```



In examining the adjustments made to Line 40, the original PRINT@ position of 191 was translated into 223 (row 2, column 63). Running the program CENTER/M4 would cause a box to be drawn on the upper left hand corner of the screen. Manual correction of the program would not be required. Notice that PRINT TAB commands (see Line 175) are not adjusted in the case of a MODIFY, as they refer to column position only.

Because the MODEL 4 video is larger than that of the MODEL III, it is possible to "overlay" a MODEL III screen onto a portion of the MODEL 4 video. The amount of screen movement available is up to 8 rows, 16 columns. In terms of performing a program conversion, the CENTER= parameter may be used in conjunction with the MODIFY parameter, to further adjust PRINT@ positions represented by numeric constants. The default value for the CENTER= parameter is 328 (4 rows, 8 columns).

The following command will perform a conversion of the program CENTER/BAS so that the "box" will be drawn on the center of the MODEL 4 screen. The resulting feed-back output is shown below.

MOD324 CENTER/BAS CENTER1/M4:3 (M,C)

File CENTER1/M4:3

The following lines may need manual correction:

```
20 PRINT@ (0=>328)
30 PRINT@ (64=>408),PRINT@ (127=>471)
40 PRINT@ (128=>488),PRINT@ (191=>551)
50 PRINT@ (192=>568),PRINT@ (255=>631)
60 PRINT@ (256=>648),PRINT@ (319=>711)
70 PRINT@ (320=>728),PRINT@ (383=>791)
80 PRINT@ (384=>808),PRINT@ (447=>871)
90 PRINT@ (448=>888),PRINT@ (511=>951)
100 PRINT@ (512=>968),PRINT@ (575=>1031)
110 PRINT@ (576=>1048),PRINT@ (639=>1111)
120 PRINT@ (640=>1128),PRINT@ (703=>1191)
130 PRINT@ (704=>1208),PRINT@ (767=>1271)
140 PRINT@ (768=>1288),PRINT@ (831=>1351)
150 PRINT@ (832=>1368),PRINT@ (895=>1431)
170 PRINT@ (896=>1448)
175 PRINT@ (960=>1528),TAB (15=>23)
180 PRINT@ (473=>913)
210 PRINT@ (473=>913)
```

In examining Line 40, the original PRINT@ position of 191 was translated into 551. The MODIFY value of 223 was first obtained. Then, the CENTER value of 328 was added in, to obtain the final result. Running the program CENTER1/M4 would cause a box to be drawn in the center of the screen (the upper left corner of the box is positioned at row 4, column 8). Manual correction of the program would not be required. Notice that PRINT TAB commands (see Line 175) are adjusted in the case of a CENTER, as movement of the entire screen affects column positioning. The value that will be added to numeric constants in PRINT TAB statements is the column offset (in this example, 8). If zero is used as a column offset (i.e. if CENTER=80, 160, 240, etc.), PRINT TABs will not be adjusted by CENTER.

When using the CENTER= parameter, the MODIFY parameter must also be specified for any adjustments to occur. Although any value may be used with CENTER=, some values (e.g. CENTER=99) will produce undesirable results. Offsets of more than 8 rows and/or 16 columns should be avoided. The following table lists the CENTER= value ranges that make the most practical sense.

CENTER= Range	Row Offset
0-16	0
80-96	1
160-176	2
240-256	3
320-336	4
400-416	5
480-496	6
560-576	7
640-656	8

### Miscellaneous "Feedback" Information

When PRINT@ and PRINT TAB statements utilize numeric expressions as print position values, adjustments to the positioning values will not be made. However, the feedback associated with such commands will indicate that the print positioning value is a numeric expression. Consider the following MODEL III program (CNTLOOP/BAS) which will draw a box on the video via a FOR-NEXT loop.

```

5 CLEAR 2000
10 CLS
20 PRINT@0,CHR$(151);STRING$(62,131);CHR$(171)
25 FORL=1TO13:A1=L*64:PRINT@A1,CHR$(149):PRINT@A1+63,CHR$(170):NEXTL
170 PRINT@896,CHR$(181);STRING$(62,176);CHR$(186);
172 MC$="Center of Box":MB$="Press Any Key to end this Program"
174 M1=LEN(MC$):M2=LEN(MB$):CM=7*64+((64-M1)/2)
175 PRINT@960,"";TAB((64-M2)/2);MB$;
180 PRINT@CM,MC$;
190 I$=INKEY$:IFI$<>""THENEND
200 FORL=1TO30:NEXTL
210 PRINT@CM,STRING$(M1,32);
220 I$=INKEY$:IFI$<>""THENEND
230 FORL=1TO20:NEXTL:GOTO180

```

The following command can be used to convert this program, with the resulting feedback output shown below.

MOD324 CNTLOOP/BAS CNTLOOP/M4:3 (M,C)

File CNTLOOP/M4:3

The following lines may need manual correction:

```

20 PRINT@(0=>328)
25 PRINT@(EXP),PRINT@(EXP)
170 PRINT@(896=>1448)
175 PRINT@(960=>1528),TAB(EXP)
180 PRINT@(EXP)
210 PRINT@(EXP)

```

Notice that an adjustment did occur in Line 20. However, in Line 25 the print position was specified as a numeric expression. In this case, an adjustment is not made to Line 25 in the output filespec (converted program). Rather, the feedback message associated with the PRINT@ statement indicates that an expression (EXP) follows the PRINT@. PRINT@(EXP) will always be displayed (regardless of the conversion parameters specified) when a numeric expression follows a PRINT@ statement.



The same type of feedback will occur with PRINT TAB statements. This will happen when a column offset is dictated by the CENTER parameter, and a numeric expression denotes the tab position (see Line 175).

Due to the expansion that takes place during a program conversion (e.g. spaces being inserted), it may be necessary for MOD324 to truncate a program line. Line truncation is done so that the resulting program file may be loaded into memory by MODEL 4 BASIC. When a line is truncated, as much of the line as possible is stored in the output program file, and a feedback message shows the part of the line that was truncated.

As an example, assume that the following line exists in a MODEL III program file.

```
10 FORLL=1TO10:FORLK=1TO20:FORLP=1TO30:LPRINTTAB(20)"This is an example of a
    converted line being too long":LPRINTTAB(20)"The value of LL is";LL:
    LPRINTTAB(20)"The value of lk is";LK:LPRINTTAB(20)"The value of LP is";
    LP:NEXTLP:NEXTLK:NEXTLL:PRINTTAB(20)"Done"
```

Consider the results of performing a conversion of this line, as shown below (shown first is the line as it would be saved to the output filespec, followed by the feedback message that would be generated).

```
10 FOR LL=1 TO 10:FOR LK=1 TO 20:FOR LP=1 TO 30:LPRINT TAB(20)"This is an
    example of a converted line being too long":LPRINT TAB(20)"The value of
    LL is";LL:LPRINT TAB(20)"The value of lk is";LK:LPRINT TAB(20)"The value
    of LP is";LP:NEXT LP:NEXT LK:NEX
```

The following lines may need manual correction:

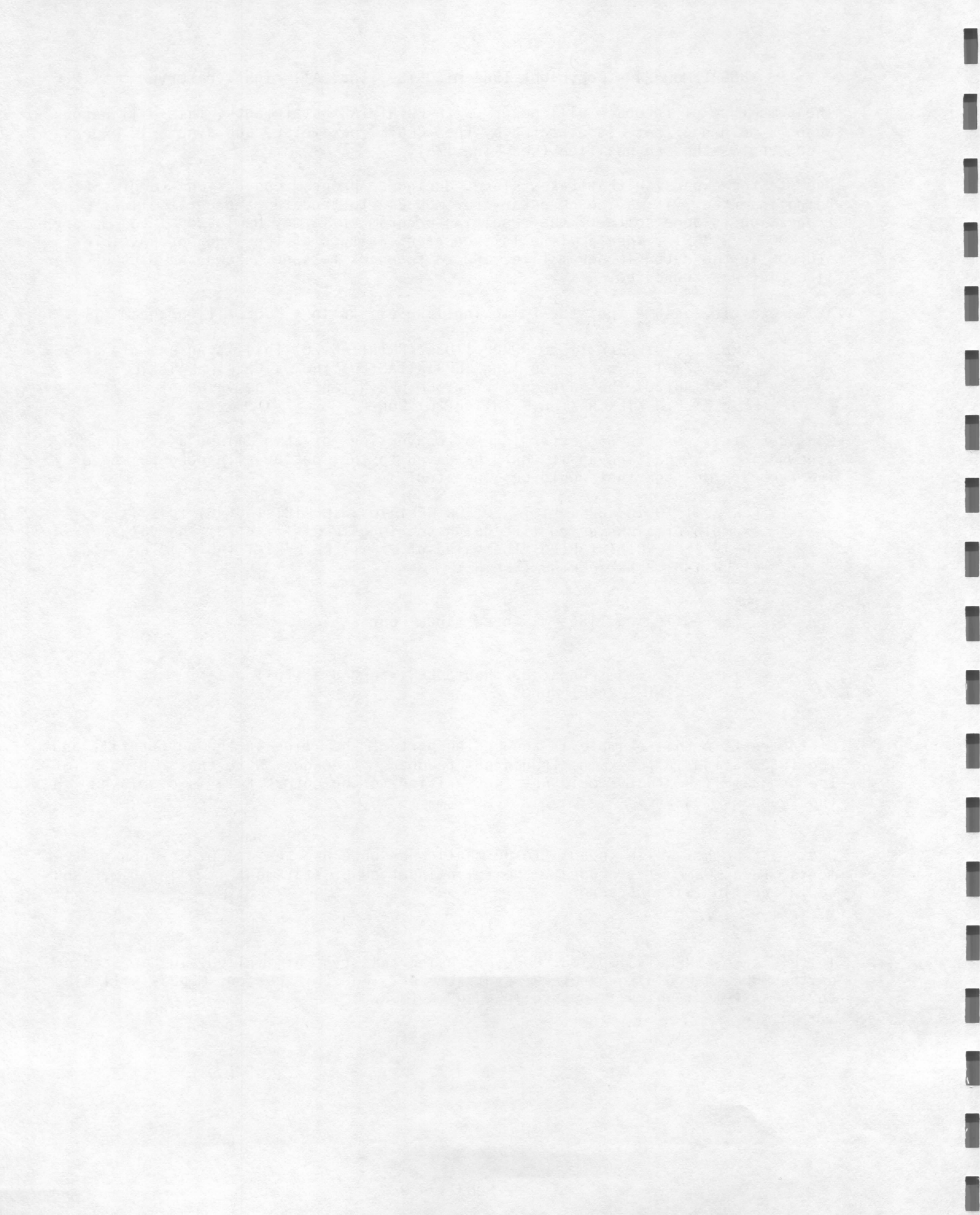
```
10    TAB
10    - Line truncated, should be extended as follows:
T LL:PRINT TAB(20)"Done"
```

Of interest in this example is the ending part of the line in the output file and the information in the "Line Truncated" feedback message. Note that any part of the program line that could not get written to the output file is displayed in the feedback message.

One last point which needs to be mentioned concerns the use of IF-THEN statements. In a MODEL III program, the following type of statement is allowable, and would function without error.

```
IF A=1 A=2
```

In this case, THEN is implied. However, using this type of implied THEN statement on the MODEL 4 would generate a syntax error. For this reason, MOD324 will flag any IF statement which is not followed by a THEN.





# F E D   I I   U S E R   M A N U A L

## Table of the Contents

FED II Overview.....	page 1
Entering FED II.....	page 2
FED II Displays.....	page 3
Command Listing.....	page 4
Cursor Movement.....	page 5
Record Manipulation.....	page 6
Record Modification.....	page 7
Control Commands.....	page 8
Output Commands.....	page 8
Search Commands.....	page 9
Load Module Commands.....	page 10
Disk Mode.....	page 11
Examples of FED II usage.....	page 12
Disk I/O Errors.....	page 14
FED II Load Errors.....	page 14
WARRANTY INFORMATION.....	page 15
Appendix One (Load Module Format).....	page 16
Appendix Two (Disk & File Structure).....	page 19

FED II User Manual  
Copyright © 1983 By Logical Systems, Incorporated  
All Rights Reserved

FED II version 5.1  
Copyright © 1983 By Logical Systems, Incorporated  
All Rights Reserved

FED II version 6.0  
Copyright © 1983 By Logical Systems, Incorporated  
All Rights Reserved





# FED II - THE LDOS FILE EDITOR

## Utility Overview

FED II allows the user to access a disk file to display, disassemble, and edit that file. It is a screen-oriented File Editor to be used with an LDOS compatible operating system. Its wide range of capabilities make it an excellent tool for the advanced user. Its simplicity makes it easy to use for the novice.

FED II's main features are as follows:

- 1) Substitution editing capabilities are supported. The user can easily position to any byte in any given record. Hexadecimal and ASCII modification are available. Direct disk patching becomes a simple matter with FED II. Small changes in files can be made quickly. With FED II, there is no need to reassemble large source files merely to change one byte.
- 2) FED II allows record advance, backspace, and absolute positioning. Paging back and forth through the file is accomplished at a keystroke. The user does not need to know any diskette information (such as density, number of sides, number of sectors per gran, etc.). The required information necessary to start FED II is the file name.
- 3) ASCII, literal text or hex string searches are easily performed. A repeat command exists to position to subsequent occurrences of the same string. FED II searches the entire file, not just the current record. It searches for text or ASCII strings up to 16 characters in length. Searching for Hex strings of up to 6 bytes in length is supported.
- 4) Mapping of machine language (/CMD) files with loader code blocking and Z-80 disassembly is available for LDOS load module format files. The user can page through each load block (forward or backward), or position to the byte in the file which loads at a specific address. It is also possible to position to the next Z-80 instruction or position to the address referenced by the current instruction. These features allow the user to step through and examine machine language routines within a file. Direct patches are made quickly and easily.
- 5) Complete listing of a file, individual records, and disassembly output of load module files to a printer are supported.
- 6) FED II includes a standard 256 byte display mode, and FED II 6.0 includes a display for files with record lengths other than 256. Under 5.1, a screen mode is available to display additional information not available in the 256 byte display mode due to lack of video space.
- 7) A Disk Mode is also available to work with an entire disk (at the cylinder/sector level).

Information on disk organization, file structure, and load module format files can be found in the Appendix of this manual.

Throughout this manual, a character or word between vertical bars is used to represent a keyboard key. Thus the symbol, |ENTER|, refers to the keyboard key marked ENTER and not a five letter word. |P| means the "P" key etc.

## ENTERING FED II

To enter FED II simply type

FEDII|ENTER|  
or  
LSFEDII|ENTER|

at the DOS Ready prompt. If a load error occurs, refer to the FED II Load Errors section. Once FED II has loaded, a prompt will appear for the filespec. Answer this prompt by giving the filespec or drive you wish to examine/modify. The filespec must be entered using the following syntax:

**\*Filename/ext.password:drivespec,lrl|ENTER|**

Note: The leading asterisk is an optional parameter (normally not used) which is explained later. The LRL parameter is available only under FED II 6.0

The **filename** may consist of up to 8 alphanumeric characters, the first of which must be alphabetic. All filespecs must contain at least the filename. The **extension** may consist of up to 3 alphanumeric characters. Like filenames, the first character must be alphabetic. The extension is optional. The default extension for all filespecs is /CMD. To enter a file which has no extension, follow the filename with a slash "/". The optional password has the same form as a filename. The **password** is necessary only if the protection level is EXEC or higher. If the file has READ access, then FED II will not allow writing to the file unless the update/owner password was given. The **drivespec** is a colon followed by a number between 0 and 7 which is a working drive number. This is an optional field. If no drivespec is entered, all active drives in the system will be searched for the filespec. The first match found will be used. If the drivespec is used by itself, the entire disk will be treated as a file (see Disk Mode). LRL is a number ranging from 1 through 256. Like the drivespec, the LRL is optional. If no LRL is specified, the default value will be 256. To exit FED II at this point rather than entering a filespec, press the |BREAK| key, and control will return to the DOS level. If an illegal or improper filespec is given, the appropriate error message will appear, and the filespec prompt will re-display.

Hexadecimal notation (X'nn') will be used to represent the current record number and relative byte number. After a valid filespec has been given, record X'0000' will appear on the screen, and be resident in the "edit buffer". The term "edit buffer" will refer to the record of the file currently in the computer's memory which is simultaneously being displayed. The edit buffer (also referred to as current record) will contain one record (1-256 bytes) at any given time. There will be two cursors flashing within the record (one cursor will be in the "ASCII" portion of the screen, the other cursor will be in the "Hex" display portion). Upon initially accessing a file, these cursors will be positioned over relative byte X'00' of record X'0000'. Throughout this documentation, the term "relative byte" will be used, and will indicate the byte number (X'00'-X'FF') relative to the beginning of the current record.

There will be an input cursor located on the lower portion of the screen following the message "Command". This will be referred to as the "command buffer", and will be used to pass commands to FED II.

Additional information shown on the screen will be the current record number, filespec, relative byte within the sector, etc. The following sample displays show where this information will be presented.

It is advised that when using FED II, the |BREAK| key should always remain enabled. The |BREAK| key is necessary to abort any operation and to terminate others.



Under FED II 5.1, there are two display modes: a full 256 byte mode and a 128 byte window mode.

### FED II 5.1 (128 Byte Display Mode)

0123456789ABCDEF BYTE 0001 0203 0405 0607 0809 0A0B 0C0D 0E0F

```

..FED ..Copyri <00> 0506 4645 4420 2020 1F17 436F 7079 7269
ght 1981-3 by LS <10> 6768 7420 3139 3831 2D33 2062 7920 4C53
I...Y*.e....ew.. <20> 4901 02D6 592A 0365 7EC9 2A03 6577 CD8D
h..h..h.00102030 <30> 68CD 0068 CD19 68C9 3030 3130 3230 3330
405060708090A0B0 <40> 3430 3530 3630 3730 3830 3930 4130 4230
C0D0E0F0          <50> 4330 4430 4530 4630 2020 2020 2020 2020
                   <60> 2020 2020 2020 2020 2020 2020 2020 2020
                   <70> 2020 2020 2020 2020 2020 2020 2020 2020

```

Record X'0000' Byte X'2A' => X'2A' = 0010 1010 = 42

Load Address = X'59DB' LD HL,(6503H)

FED51/CMD:0 Command:

### FED II 5.1 (256 byte mode)

```

..FED ..Copyri | 00> 0506 4645 4420 2020 1F17 436F 7079 7269 | 0 | F
ght 1981-3 by LS | 10> 6768 7420 3139 3831 2D33 2062 7920 4C53 | 0 | E
I...Y*.e....ew.. | 20> 4901 02D6 592A 0365 7EC9 2A03 6577 CD8D | 0 | D
h..h..h.00102030 | 30> 68CD 0068 CD19 68C9 3030 3130 3230 3330 | 0 | 5
405060708090A0B0 | 40> 3430 3530 3630 3730 3830 3930 4130 4230 | 0 | 1
C0D0E0F0          | 50> 4330 4430 4530 4630 2020 2020 2020 2020 | 0 | /
                   | 60> 2020 2020 2020 2020 2020 2020 2020 2020 | 0 | C
                   | 70> 2020 2020 2020 2020 2020 2020 2020 2020 | 0 | M
                   | 80> 2020 2020 2020 2020 2020 2020 2020 2020 | 0 | D
                   | 90> 2020 2020 2020 2020 2020 2020 2020 2020 | 0 | :
                   | A0> 2020 2020 2020 2020 2020 2020 2020 2020 | 0 | 0
                   | B0> 2020 2020 2020 2046 4544 3220 2D20 4669 | 0 |
FED2 - Fi | C0> 6C65 2045 6469 746F 7220 5665 7273 696F | 0 |
le Editor Versio | D0> 6E20 312E 0341 2020 0343 6F70 7972 6967 | 0 |
n 1..A .Copyrig | E0> 6874 2031 3938 312D 3320 6279 204C 6F67 | 0 | >2A
ht 1981-3 by Log | F0> 6963 616C 2053 7973 7465 6D73 2049 6E63 | 0 | C:
ical Systems Inc

```

# Display mode under FED II for 6.0

0123456789ABCDEF	BYTE	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
.....(.....KI	<00>	00	FE	14	01	00	00	28	10	05	C3	08	00	00	00	4B	49
.....DO.<....PR	<10>	07	D0	0B	00	00	00	44	4F	06	3C	0E	00	00	00	50	52
.....SI.....SO	<20>	15	08	02	0D	00	00	53	49	17	10	02	0F	00	00	53	4F
.....JL.g.=..X	<30>	0A	00	00	0A	00	00	4A	4C	CD	67	02	3D	20	0C	CD	58
..g.w#...=(..g.	<40>	02	CD	67	02	77	23	10	F9	18	EE	3D	28	0B	CD	67	02
G.g.....g.G.g.o	<50>	47	CD	67	02	10	FB	18	E0	CD	67	02	47	CD	67	02	6F
..g.g...,...tC.	<60>	05	CD	67	02	67	05	C9	D9	2C	20	0D	E5	CD	74	43	E1
.(.....!.	<70>	1C	7B	D6	12	20	02	5F	14	7E	D9	C9	01	88	0F	21	9B
...A..+.....cPV.	<80>	02	7E	ED	41	D3	89	2B	05	F2	81	02	C9	63	50	56	08
.....e.....	<90>	18	00	18	18	00	09	65	09	00	00	00	00	00	00	00	00
.....	<A0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<B0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<C0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<D0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<E0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<F0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

BOOT/SYS:0      Record X'0000'      Byte X'84' => X'D3' = 1101 0011 = 211

Command:

## FED II LIBRARY

```

|A|      Enter ASCII modification mode
|B|      Position to the Beginning of the file
|C|ENTER| Clear record with zeroes
|D|ENTER| Disassemble file to printer
|E|      Position to the End of the file
|F|      Enter Find mode and:
|A|      find ASCII string
|H|      find Hexadecimal string
|T|      find Text string
|L|      find Load address
|G|      Go to the next occurrence of last search
|H|      Enter Hex modify mode
|I|      Position to next Z-80 Instruction
|J|      Jump to current instruction reference
|L|ENTER| List disk file to printer
|M|      Toggle between 128 and 256 byte display Modes (5.1 only)
|N|ENTER| Enter a New file
|P|ENTER| Print current record in edit buffer
|R|      Position to Record
|S|ENTER| Save current record in edit buffer
|U|ENTER| Update file's directory entry
|X|ENTER| EXit FED II and return to LDOS Ready
|BREAK|  Cancel current FED II command
|ENTER|  Display FED II instruction set (Menu)
|:| +|   Advance one record in the file
|-| -=|  Backup one record in the file
|<|      Position to previous load block
|>|      Position to next load block
|/|      Toggle between Directory entry and HIT position

```



## NOTE

FED II is an advanced utility, giving the user an opportunity to accomplish tasks not easily performed by other means. FED II can also be a hazard to the inexperienced or uninformed user. It is strongly recommended that all editing be done on a BACKUP copy of the original file or disk whenever possible.

### Cursor Movement Commands

@  nn	Position cursor to relative byte X'nn'
Lt arrow	Move cursor left one byte
Rt arrow	Move cursor right one byte
Up arrow	Move cursor up one line
Dn arrow	Move cursor down one line
SHIFT  Up arrow	Position cursor to relative byte X'00' of the current record
SHIFT  Rt arrow	Position cursor to end of the line
SHIFT  Lt arrow	Position cursor to start of the line

The @ positioning command requires a hex byte consisting of two hex digits. This may be any number X'0' through X'FF', however, if a single digit is used (no leading zero), the |ENTER| key must be pressed in order to execute the command. This is because the |@| command expects two digits. If two digits are used, the command will execute immediately after the second one is typed.

This arrangement is used throughout FED II whenever information must be supplied in addition to the command key. Merely remember to press |ENTER| if nothing occurs after typing in a command. This is especially true for the strings used in all of the FIND subcommands.

The arrows may be used from within either the ASCII or Hex modification modes to position within the buffer. The "@" sign will not be accepted during either modify mode.

The cursor movement commands will not wrap into either a prior or a subsequent record. To switch records, use the record manipulation commands. No cursor movement will occur if an attempt to violate buffer boundaries results.

## RECORD MANIPULATION COMMANDS - FILE MODE

- |+|** Advance one record sequentially in the file. For example, if the current record is X'000C', after pressing |+|, record X'000D' would be displayed (provided that it exists). An "\*" will be displayed adjacent to the record number when positioned to the last record in a file. Issuing the |+| command will not change the position of the relative byte cursors. A "+" will be shown in the command buffer to indicate forward motion in the file.
- |-|** Back up one record sequentially in the file. If the current record is X'0087', after pressing |-|, record X'0086' would be displayed. Issuing the |-| command does not change the position of the relative byte cursors. The |-| command will be ignored if it is issued when record X'0000' is being displayed. A "-" will be shown in the command buffer to indicate retrograde motion in the file.
- |B|** Position to the beginning of the file (record X'0000') and position cursors to relative byte X'00'.
- |E|** Position to the end of the file. An "\*" will appear adjacent to the record number, indicating that the record being displayed is the last record in the file. If the file has an LRL of 256, then the relative byte cursors will be positioned on the last byte in the file which is often referred to as the "End-Of-File offset byte". (Note that this is not necessarily relative byte X'FF'.) For LRLs other than 256, the cursors will be positioned at the last byte in the record. Any modifications made to bytes beyond the EOF offset byte are usually superfluous.
- |R|nnnn** Position to Record X'nnnn', provided record X'nnnn' exists in the file. If the record does not exist the request will be ignored. After entering |R|, the prompt Record X' ' will appear in the command buffer. The input for the record number will be taken within the single quotes. Hex digits (0-F) must be entered. Any other characters will be ignored. |BREAK| will cancel this command. The user may enter the record number without using the standard four digit (X'nnnn') format. Simply type in the record number and press |ENTER|. For example, if the desired record number is X'0021', type |R||2||1||ENTER|. To position to record X'0007', type |R||7||ENTER|. The position of the relative byte cursors will remain unchanged after positioning to the new record.
- |/|** This command either positions from a byte in the HIT (Hash Index Table) to a directory entry or from a directory entry, back to the HIT. If the cursor is positioned somewhere in the HIT, |/| would reposition to the directory entry corresponding to that HIT position. If positioned in a directory entry, |/| would reposition to the byte in the HIT corresponding to that entry. This command is only applicable while in DIR/SYS (file mode) or in the directory cylinder (disk mode). It serves no purpose elsewhere and does not function in any other file or any other cylinder. For more information, consult your DOS manual on DIRECTORY structure.



## FED II MODIFICATION COMMANDS

- |A|** Enters the ASCII Modification Mode. In this mode, modifications can be made directly in ASCII. Any character that can be generated from the keyboard (with the exceptions of the |BREAK| key and the arrow keys) can be directly entered into the edit buffer. Modifications can be made by positioning the cursor over the bytes to be changed. After the |A| command is issued, the cursors will become larger and the command buffer will display "ASCII Modify". (In the 256 mode of 5.1, an "A" will display.) From this point on, any characters entered will be taken as modifications to the bytes in the edit buffer. After a character is entered, the cursors will position to the next character in the edit buffer. If the cursors are positioned at the last character in the edit buffer, then no advance will occur. The arrow keys may be used to position the cursor without altering the buffer contents. Any changes made to the buffer WILL NOT automatically write to the file. In order to save changes to the file, see the |S|ave command. To exit the ASCII modify mode, press the |BREAK| key.
- |H|** Enters the Hexadecimal Modification Mode. In this mode, modifications to bytes in the edit buffer are accomplished by typing hexadecimal digits. After the |H| command is issued, the cursors will become larger and the command buffer will display "Hex Modify" ("H" in the 256 mode of 5.1). From this point on, hexadecimal digits (0-F) must be entered to modify bytes in the buffer. Note that since a single byte is represented by two hex digits, hex modify edits one nibble (half a byte) at a time. The arrow keys may also be used to position the cursors for additional editing. To exit the Hex modify mode, press the |BREAK| key. Like the ASCII Modification Mode, no changes are automatically made to the file. To make the modifications to the file, see the |S|ave command.
- |C| |ENTER|** Clears the buffer contents from the cursor position to the end of the buffer by filling it with X'00'. Some files have erroneous or random information past the end-of-file offset byte in the last record. The clear command can be used to overwrite the remainder of the buffer with zeros to facilitate viewing. Again, since none of the edits perform an automatic write to disk, |S| is necessary to save the buffer contents.
- |S| |ENTER|** Save the contents of the current edit buffer to disk. The current record will overwrite the contents of the disk record. Although the changes are made, neither the date nor the mod flag of the file in the directory record will be changed.
- |U| |ENTER|** Updates the directory entry of the current file being edited. The mod flag will be set, and the date (if maintained) will be updated. This is normally not done by FED II even if the file has been altered via the |S|ave command. Update only works in the file mode.

### FED II Control Commands

- |N||ENTER| Causes a prompt for a new filespec. FED II will clear the screen, print its sign-on message, and prompt the user for a new filespec to be edited. Note that strings are saved so that |G| will work on the new file without re-entering a search criterion.
- |X||ENTER| Exit to operating system. Any changes made to the current record buffer, and not written to disk with the |S| command will be lost.
- |M| Switches between the 128 and 256 byte display modes. This command only exists in FED II 5.1.
- |ENTER| The |ENTER| key is used as a confirmation to complete most commands that result in significant alteration of the current record. |ENTER| alone will display a menu containing most of the FED II commands, and a brief description of their use. Pressing |ENTER| at the menu page will return to the display mode.
- |BREAK| The |BREAK| key is used to abort a FED II command in progress.

### FED II OUTPUT COMMANDS

- |P||ENTER| Print the current buffer contents, to a printer. If the printer is not available, the error message "Printer Not Ready" will display. Pressing |ENTER| at the error will attempt to print again. Pressing |BREAK| will abort the operation. Under LDOS 6.0, other error messages are possible if the printer is routed or linked to a disk file or a device.
- |L||ENTER| List the file to a line printer starting with the current record. The record length must be 256. Since FED II does its own pagination, it is suggested that no printer filters involved with line counting be used in conjunction with this command. The listing will terminate when the end of the file is encountered, or when |BREAK| is pressed. Error handling works exactly as with the |P| command.
- |D| Output disassembly of a Load Module File starting with the current instruction to a line printer. If the cursor is positioned in loader code, the first instruction following will be used. The listing will terminate when the end of the load module is encountered, or when |BREAK| is pressed. Unlike the |L| command, there is no pagination on the disassembly output. Error handling is identical to the |P| command.



## FED II SEARCH COMMANDS

- |F| Enters the FIND mode. A "FIND" prompt will appear in the command line ("F" in 256 mode in 5.1). A subcommand declaring the type of search must be entered. After entering the subcommand, an appropriate prompt will display followed by a blank space in quote marks. (The letters A, H, T, or L will display in the 256 mode of 5.1). The following are the four FIND sub-commands:
- |A| Find ASCII "string". This is a literal search for the exact ASCII characters entered. "string" is a group of from one to sixteen ASCII characters. The only ASCII characters that can't be generated are the |ENTER|, |BREAK| and |BACKSPACE| keys. If less than 16 characters are typed, the |ENTER| key must be pressed to initiate the search.
- |H| Find Hexadecimal "string". This is a literal search for the exact hexadecimal bytes entered. "string" is a group of up to 12 hex digits (6 bytes). Only the valid hex characters (0-F) will be accepted.
- |T| Find Text "string". This is a search for ASCII characters that ignores differences in upper or lower case. The same restrictions which apply to |A| apply to |T|.
- |L| Find Load Address X'nnnn'. This is a search for the byte in the file which loads at memory location X'nnnn'.
- |G| Go to the next occurrence of the last searched string or load address. In order to do the same search again, it is NECESSARY to use this command. It simply looks for the last item specified again. It also "memorizes" the last search criterion as long as FED II is active. This means that searches through different files for the same string are possible without re-entry of the string. The only exception to this is an attempted |F||L| when switching to a non-load module format file. Obviously the string is useless in that case. If it is not obvious then read the appendix. If it is still not obvious just take our word that it makes no sense.

In FED II 5.1, using the 256 byte mode, searches are limited to 6 characters in length (3 bytes if Hex) due to screen formatting.

Pressing |BREAK| at any time during the input sequence or actual search, will cause FED II to display the record which was current before the search started. Any changes made to the current edit buffer and not saved will be lost during a search. The |BACKSPACE| key may be used to correct any mistakes made during input.

During a search, the record number being searched will be displayed. If the string or load address is not found, the appropriate message will be displayed. Control will return back to the position in the file before the search. Note : A search starts at the byte following the current position. If the cursor were positioned to relative byte X'FF' of record X'0012', the search would start at relative byte X'00' of record X'0013'. For load module format files, only data found in object code blocks will be used in the search. Characters in header, comment, filename or any other blocks will be ignored. It is also assumed that object code blocks load contiguously in memory. In order to search a Load Module File and include load blocks, precede the filespec prompt (the very first thing FED II asks for) with an asterisk (\*).

## FED II Load Module Commands

- |I| Position to the next Z-80 instruction. For example, if the cursor is positioned at a CALL instruction, and the |I| command was used, the cursors would be positioned 3 bytes after the X'CD' opcode. If the instruction spans a load block, an additional 4 bytes will be skipped. Note: Using the |I| command may position to the next record in the file. If this should happen, any changes made to the edit buffer which were not saved will be lost. If the cursor is positioned in load code, |I| will position to the first valid instruction found. Note that opcodes out of sequence will be just as readily disassembled. Make sure the "logic thread" being followed is the correct one.
- |J| Position to Z-80 instruction reference. This command positions to the address operand of the current instruction. For example, if the current instruction was a JP 67A9, issuing a |J| would attempt to position to the byte which would load at address X'67A9'. If the address is not located in the file, the error message "Load Address not Found" will be displayed, and the original cursor position will be restored. The |J| command may be used for any instruction referring to an absolute address or relative branch (CALL, JP, JR, LD, DJNZ), whether conditional or unconditional. If the cursor is either at the end of the module or with the transfer address block, a |J| will locate the transfer address. Any address used by |J| must be a 16 bit address or a JR offset. Branches such as eight bit loads, JP (HL) or RST will not work. For example, the instruction LD A,5 will not attempt to locate X'0005', LD HL,6060 will, however, attempt to locate X'6060'. Note: any changes made to the edit buffer prior to using the |J| command will be lost if the current record is left, unless saved with the |S| command.
- |>| |<| Positions the cursors to the next/previous loader block (X'01', X'02', X'03', X'05', X'07', X'10', X'1F') of a Load Module File. This feature was designed to allow the user to trace through machine language files quickly. Encountering a X'02' will terminate a trace. For more information on "Type" bytes, refer to the appendix on LOAD MODULE FORMAT FILES.



## DISK MODE

Occasionally it is desirable to work with an entire disk as opposed to a single file. For this purpose, the FED II disk mode makes it possible to treat the entire disk as a file. Most of the commands available in the file mode are also available in the disk mode. To enter the disk mode, simply give the drivespec (colon followed by a number between 0 and 7) at the "Filespec:" prompt.

Since the entire disk is treated as a file, positioning to specific cylinders and sectors is accomplished by specifying the record number.

The following commands illustrate the difference. More information may be obtained from the appendix on disk organization.

- |R| ccss      Position to Record X'ccss' on disk. The Record number consists of the cylinder number (cc) and sector number (ss). For example, positioning to cylinder X'34', sector X'05' (record X'3405'), would be accomplished by typing |R||3||4||0||5|. Pressing |ENTER| following the record number is required only for requests consisting of less than four digits. To position to Record X'030A', type |R||3||0||A||ENTER|. To position to Record X'0004' type |R||4||ENTER|.
- |B|            Position to Beginning of Disk (Cylinder 0, Sector 0). After issuing a |B|, the current Record number would be X'0000'.
- |E|            Position to End of Disk. The actual record number would depend on the type of disk. A single-sided double density 40-track diskette would have an ending record of X'2711' (Cylinder X'27', Sector X'11').
- |+|            Position to next record. If positioned at record X'1405', (Cylinder X'14', Sector X'05'), after pressing |+|, record X'1406' (Cylinder X'14', Sector X'06') would be displayed. If the |+| is used when positioned at the ending sector number of a cylinder, sector 0 of the the next cylinder would be displayed. For example, a double-density, double-sided, five-inch diskette has 36 sectors per cylinder (numbered from X'00' - X'23'). If the current record is X'0223', after issuing |+|, record X'0300' would be displayed.
- |-|            Position to previous record. If positioned at record X'1405', (Cylinder X'14', Sector X'05'), after pressing |-|, record X'1404' (Cylinder X'14', Sector X'04') would be displayed. If the |-| is used when positioned at Sector 0 of a cylinder, the ending sector of the previous cylinder would be displayed. For example, one configuration for a 5" hard disk might use one platter for a logical drive. Each cylinder might contain 64 sectors (numbered X'00 - X'3F'). Issuing a |-| when positioned at Record X'5000' (Cylinder X'50', Sector X'00') would cause record X'4F3F' (Cylinder X'4F', Sector X'3F') to be displayed.

Another feature of the DISK MODE is current file indication. Under 5.1, this is shown in the 128 byte display mode. When positioned to a sector on a disk which is allocated to a file, the filespec along with the relative record number is displayed. With this feature, reconstruction of a damaged directory is possible.

If FED II attempts to access a disk which it cannot distinguish, the error message "Can't Log in Disk" will be displayed. Like other errors, pressing [BREAK] will cancel the current command, causing the filespec to re-prompt. Pressing [ENTER] will indicate to FED II that it should use the information in the DCT (Drive Code Table) to access that disk. If FED II cannot interface properly, it may be necessary to use DEBUG or another utility to fix the disk.

#### PRACTICAL EXAMPLES OF FED II'S USE

- 1) Change the byte which loads at address X'57CE' to an X'C9', in a file named TEST/CMD on drive 2:
  - A) To edit the file, at the Filespec prompt type:  
TEST:2|ENTER|
  - B) To position to address X'57CE', type:  
|F||L|57CE
  - C) To enter hex modify mode, press:  
|H|
  - D) Then type in the change:  
C9
  - E) To exit the hex modify mode, type:  
|BREAK|
  - F) To save the change to disk, type:  
|S||ENTER|
  - G) To exit FED II, type:  
|X||ENTER|

Note that in this case, no extension was required to access the file, because the default extension of /CMD was correct.

- 2) To null out record X'7A' of a data file named ACCOUNTS/DAT which has an update password of BOSS:
  - A) To edit the file, at the filespec prompt type:  
ACCOUNTS/DAT.BOSS|ENTER|
  - B) To position to record X'7A', type:  
|R|7A|ENTER|
  - C) To fill the buffer with zeroes, type:  
|C||ENTER|
  - D) To save the changes to disk, type:  
|S||ENTER|
  - E) To exit FED II, type:  
|X||ENTER|



- 3) To change the string "Burgers" to "Hot Dog" in a file named THEMENU/SCR on drive 6, after editing a different file:
  - A) To enter a new file, type:  
|N|ENTER|
  - B) To edit the file, type:  
THEMENU/SCR:6|ENTER|
  - C) To find the ASCII string "Burgers", type:  
|F||A|Burgers|ENTER|
  - D) To enter the ASCII modify mode, type:  
|A|ENTER|
  - E) To change the ASCII string to "Hot Dog", type:  
Hot Dog
  - F) To exit the ASCII modify mode, type:  
|BREAK|
  - G) To save the changes to disk, type:  
|S|ENTER|
  - H) To exit FED II, type:  
|X|ENTER|
- 4) To find out the name of a file on drive 0 containing the string "JOE DUDE" (in either upper or lower case):
  - A) To access the disk as a file, enter the drivespec:  
:0
  - B) To find the text string, type:  
|F||T|Joe dude|ENTER|
  - C) Under 5.1, enter the 128 byte display mode by typing:  
|M|
  - D) The filename/ext and relative record containing the string will be displayed at the lower portion of the screen.
- 5) Find the load address which contains the sequence of bytes X'45', X'22', & X'77' in a file named HELPME/DCT:
  - A) To access the file, type:  
HELPME/DCT|ENTER|
  - B) To find the hexadecimal string X'452277', type:  
|F||H|452277|ENTER|
  - C) The load address displayed refers to the first byte in the string.

## DISK I/O Errors

As with any hardware, there is always that chance of something going wrong. This could happen when reading from or writing to a disk. For some reason, some component failed to do its job. The problem could be in the disk media, the disk drive, the disk controller, or the computer. Whenever an I/O error occurs under LDOS, an error number is returned to the program that requested the disk I/O function. FED II reports the error and allows the user to decide what to do about it. There are two options available in FED II : 1) Abort the process, and resume whatever was done prior to the I/O error, 2) Ignore the error, and continue the process. Pressing [BREAK], indicates an Abort operation. Pressing [ENTER] will ignore the error, and continue the process.

## Error Examples

### Example #1

Assume that a record from a file on a write protected disk was read in. After examining the record, some changes were made to the edit buffer. Once the operator was content with the changes, the |S|ave command was issued. Almost immediately, the error message "Write Protected Disk" is displayed. To make the changes, simply take off the write protect tab and re-issue the |S|ave command.

### Example #2

While paging through a file, the error message "Parity Error During Read" is displayed. The edit buffer contains the record which had the error, however the integrity of the data is doubtful. At this point, the edit buffer could be modified and followed by a |S|ave attempt.

For a detailed description of I/O errors, refer to the Operating System manual.

## FED II Load Errors

Two rare errors need to be examined. If while attempting to execute FED II, the error message "Insufficient Memory to Load FED" appears, one of two undesirable events has occurred. 1 - the amount of memory left in the system is not enough to execute the program or 2 - Loading FED has overwritten reserved memory (HIGH\$). Because of the danger of #2, the solution is to re-boot the system. Re-enter FED II only if more free memory is available.

The second rare error is that a file is so large and the amount of free memory so sparse that FED II runs out of memory to map a file into load blocks. No error message will be issued but the current file will NOT be mapped. This is exactly like typing "\*" for the first character of the filespec. This error is not fatal but inconvenient.

Both errors occur rarely because the amount of memory which must be reserved to cause either problem is unrealistic in normal use. However, in the interest of complete error trapping, these accommodations have been made.



## WARRANTY

All products sold by Logical Systems Incorporated, hereinafter referred to as LSI, grant the user certain customer support privileges. This support shall be limited to the privilege of having the master diskette updated as often as desired for the current update fee. This is limited to updates within the current Series of the program. LSI will also provide a lifetime warranty on the physical diskette media of the original serialized master diskette. If the diskette media physically fails to retain the original program, replacement media will be provided at no charge. *This does not* include media that has been damaged in shipment from the user to LSI, or media that has been damaged by the user or their equipment. To receive this support, the user **MUST** fill out and return a specific registration card pertaining to the product, within 30 days of purchase. Should a user find a valid error in the program and clearly define it in writing to LSI, every effort will be made to correct the error. All support shall apply only to registered owners.

Logical Systems Incorporated and its associates assume no liability whatsoever, with regard to the reliability and/or fitness of their products. All data entrusted to these programs and the computer that it is operating on are the sole responsibility of the user. Under no circumstances will LSI or its associates be held liable for the loss of TIME, DATA, PROGRAMS or for any consequential damages incurred by the user.

This manual, as well as the accompanying programs and data files, are Copyrighted © by Logical Systems, Incorporated, all rights RESERVED. Reproduction and/or distribution by any means, is hereby forbidden except by written consent.

For additional information, please contact:

Logical Systems Incorporated  
P.O. Box 23956  
8970 N. 55th Street  
Milwaukee, Wisconsin 53223  
(414) 355-5454

## Appendix One

### LOAD MODULE FORMAT FILES

One of the most frequent tasks requested of the operating system is the system's own internal loader. Its function is to load machine language programs from load module files into memory. Everything from application programs such as FED II and TBA, to utilities like FORMAT and BACKUP, and even system files all are loaded via the system loader. All programs do not load at the same address, they seldom have the same length, and they rarely have the same execution address. Therefore, a special format was established to provide this variable information to the system loader. This is now called Load Module Format. When an LDOS compatible assembler, such as EDAS, writes an object file program (/CMD) to disk, it uses this specific format. Data is written in blocks, not necessarily contiguous.

Each block consists of:

- 1) 1 byte indicating the Type of block
- 2) 1 byte indicating the Length of this block
- 3) Data pertinent to the particular block type

The blocks are organized sequentially, the length of the block defined indicates the position of the next block. The system loader reads until it encounters a block type indicating that it should cease loading. The following is a list of Type bytes and functions that FED II will acknowledge.

Type Byte	Function
X'01'	Load Object block into memory
X'02'	Get Transfer Address
X'03'	Get Transfer Address (non-executable)
X'05'	Load Module Filename Header
X'07'	Patch Header Name
X'10'	Yanked X-Patch Object block
X'1F'	Comment Block

The byte following the Type byte is a length byte, which indicates the the quantity of data bytes following. The length byte has a range from X'01' to X'00' (1 to 256; note that zero is high here). A length byte of X'07' indicates seven bytes in the data field. A length byte of X'00' indicates 256 bytes of data. For Type bytes X'01' and X'10', the quantity of object code data bytes following is equivalent to the length byte minus two. This is because object code blocks contain an additional two bytes immediately following the length byte indicating the load address (See Type Byte X'01'). A length byte of X'03', indicates three bytes follow: 2 bytes for the address, and 1 byte of object code data. A length byte of X'00' indicates 256 bytes follow: 2 bytes for the address, and 254 bytes of object code data. Since the address field is always present, its length is assumed by the loader. A length byte of X'01' indicates 255 bytes (X'FF') of object code data following. A length byte of X'02' indicates 256 (X'00') bytes of object code. By subtracting two from the length byte, the actual quantity of bytes loaded can be obtained.

The data following the length byte is dependent of the type of block. There are no restrictions on what the data in the block must be. Generally, block types X'05' and X'1F' contain ASCII text, but aren't required to.



### Load Object Block - Type X'01'

This type indicates that the following data is to be loaded into memory. The two bytes following the length byte are the destination (or starting load address) of the object block. The load address is stored in LSB, MSB (Least Significant Byte followed by Most Significant Byte) format. Since two bytes of the block are used for the load address, the length byte must account for this. The easiest way to understand this is by looking at a typical block:

01 08 00 70 CD 47 95 C8 B7 D0

The length byte indicates that this block is X'08' bytes long. The address to load the data block at is X'7000' (lsb,msb format). However, only X'06' bytes are loaded into memory. This is because two bytes of this block were used to designate the load address. Therefore the actual area of RAM that will be loaded is X'7000' - X'7005'. After this block is loaded, location X'7000' will contain an X'CD', X'7001' will contain X'47' etc.

### Yanked Patch Block - Type X'10'

This type of block is used exclusively by the LDOS PATCH utility. When an X-patch is installed in a load module file, a header block and series of object blocks (X'01's) are appended to the end of the file (overwriting the last Transfer address block). When the patch is YANKed, all of the object blocks belonging to that patch are changed to X'10's so that the system loader won't load them.

### Transfer address Block - Types X'02' and X'03'

These types inform the loader where to begin execution once the entire module has been loaded into memory. Since it only takes two bytes to store an address, any bytes following the address with a length other than X'02' would be unused. In the following example:

02 02 6A 3F

The transfer address of the module would be X'3F6A' (the address is stored in lsb,msb format). Most assemblers allow the transfer address to be specified in an END statement. The only difference between the X'02' and X'03' type bytes is that the latter indicates a file that is not executable.

### Header and Comment Blocks - Types X'05', X'07', and X'1F'

These types do not actually load into memory at all. Most assemblers write an X'05' type as the first block of the file. It usually has a length of 6 bytes and the data following is most likely the first 6 characters of filename. The LDOS PATCH utility generates the X'07' type block preceding X-patch data blocks. The data contained therein is the name of the patch file. This is necessary in order to YANK the patch by that name. Assemblers most likely do not generate this type of block. This and type X'10' are used primarily by the PATCH utility. It is sometimes desirable to insert comments into a program but not have the comments resident in RAM. A block type of X'1F' indicates a comment block to the loader. Note: none of these block types specified (X'05', X'07', X'1F') have any effect on the object code.

When FED II recognizes a file as Load Module format, it reads through the file and maps every block. Once FED II has successfully mapped the file, additional information relevant to that file is now accessible. Wherever the cursors are positioned in the file, the appropriate load module information is displayed. The following are the block types and the messages that will be displayed:

X'01' - Block X'nnnn' - X'nnnn'  
X'02' - Transfer Address X'nnnn'  
X'03' - Transfer Address X'nnnn'  
X'05' - File Header Block  
X'07' - Patch Header Block  
X'10' - \*Block X'nnnn' - X'nnnn'  
X'1F' - Comment Block

When positioned at actual object code in a load block, the following will be displayed:

Load Address X'nnnn'

This is the memory location that will receive the byte at the current cursor position. An asterisk before this indicates that this byte would load at that address, except that this block has been yanked (see type X'10'). Alongside the load address will also be the mnemonic Z-80 instruction. The |L| function of the |F|ind mode allows positioning the cursors to a byte in a file which loads into memory at the specified address. The |I| command positions the cursors to the next instruction. The |J| command locates the position that the current instruction refers to (either direct or indirect). Both of these commands allow for instructions spanning load blocks or sectors. With these commands, it is possible to step through a machine language program.

When in a load module file, some information will always be displayed, regardless of the current function mode (hex modify, ASCII modify, normal). When using any modify mode in a load module file, pay VERY close attention to what is being changed. If loader codes are ever overwritten accidentally, disastrous results may occur. This might be realized the next time the file is loaded. If the message "Load file format error" appears, this means that an illegal type byte was encountered. This could have happened because the type byte itself was changed, or a length byte was changed indicating the incorrect position of the next type byte.

In some cases, it might be desirable to view a load module file as a data file. This is accomplished by typing an "\*" before the filespec at the prompt. This action will prevent the file from being mapped. Any string searching will encompass the entire file, including load blocks. For example, to hide a block of object code by changing an object block, type byte (X'01') to a yanked patch block (X'10') or comment block (X'1F'), or any other type byte. It is also possible to change the load address in an object block, or data in a comment block or patch header block. These actions are not common, but this feature is provided for advanced users.



## Appendix Two

### DISK STRUCTURE

In order to store information of any type on a disk, it must be formatted. Formatting is the process in which the disk media's magnetic surface is organized into concentric circular regions called tracks. Five inch floppy disks can be formatted anywhere from 2 to 80 tracks. Typical standards are 35, 40 and 80, depending on the drive. Some disk drives have more than one head, in which case the term "cylinder" comes into play. A cylinder is a collection of tracks grouped together as one logical unit. For example, a double headed 40 track drive has 80 total tracks, but only 40 cylinders. Each pair of adjacent tracks on opposite sides of the diskette form a cylinder. Cylinders are numbered sequentially starting at the outside edge of the diskette, which is cylinder zero. Each track is divided into smaller units called sectors. The quantity of sectors per track depends on the disk type and density. Single density 5" floppy disks have 10 sectors per track, and double density 5" floppy disks have 18 sectors per track. Some hard drives have up to 32 sectors per track and 256 sectors per cylinder. Each sector contains smaller units known as bytes. One character of data is equivalent to a byte.

### DISK FILES

After formatting and verifying the cylinders on the disk, system information must also be written to that disk. "System Information" is a collective name for two disk files - BOOT/SYS and DIR/SYS. A disk file is a collection of sectors on a disk in a specific order. A file is referred to by means of its filename, extension, and drive number. This can be abbreviated to the term "File Specification" or FILESPEC. The first portion of cylinder zero is allocated to a file called BOOT/SYS, which contains information necessary for that diskette to boot up. One full cylinder is devoted to a file called DIR/SYS also known as the directory. The directory is a collection of tables and maps used to determine anything about that disk - the disk name, password, and date of creation (specified during format), how much space is available, how much is used, what files exist, how much space they occupy and where those files are located on that disk. ANY data stored on that disk is stored in a file. Each file is made up of logical units called records, referred to by a number in the range of 0 through 65535. Each record has a fixed length between 1 and 256 bytes, the most common being 256 (the same size as a physical sector). For example, the file named BOOT/SYS contains 5 records, each with a logical record length of 256. The information such as file length, record length, dates and status are collectively called attributes of a file. To see the attributes of any file, use the DIR command with the (A) parameter.

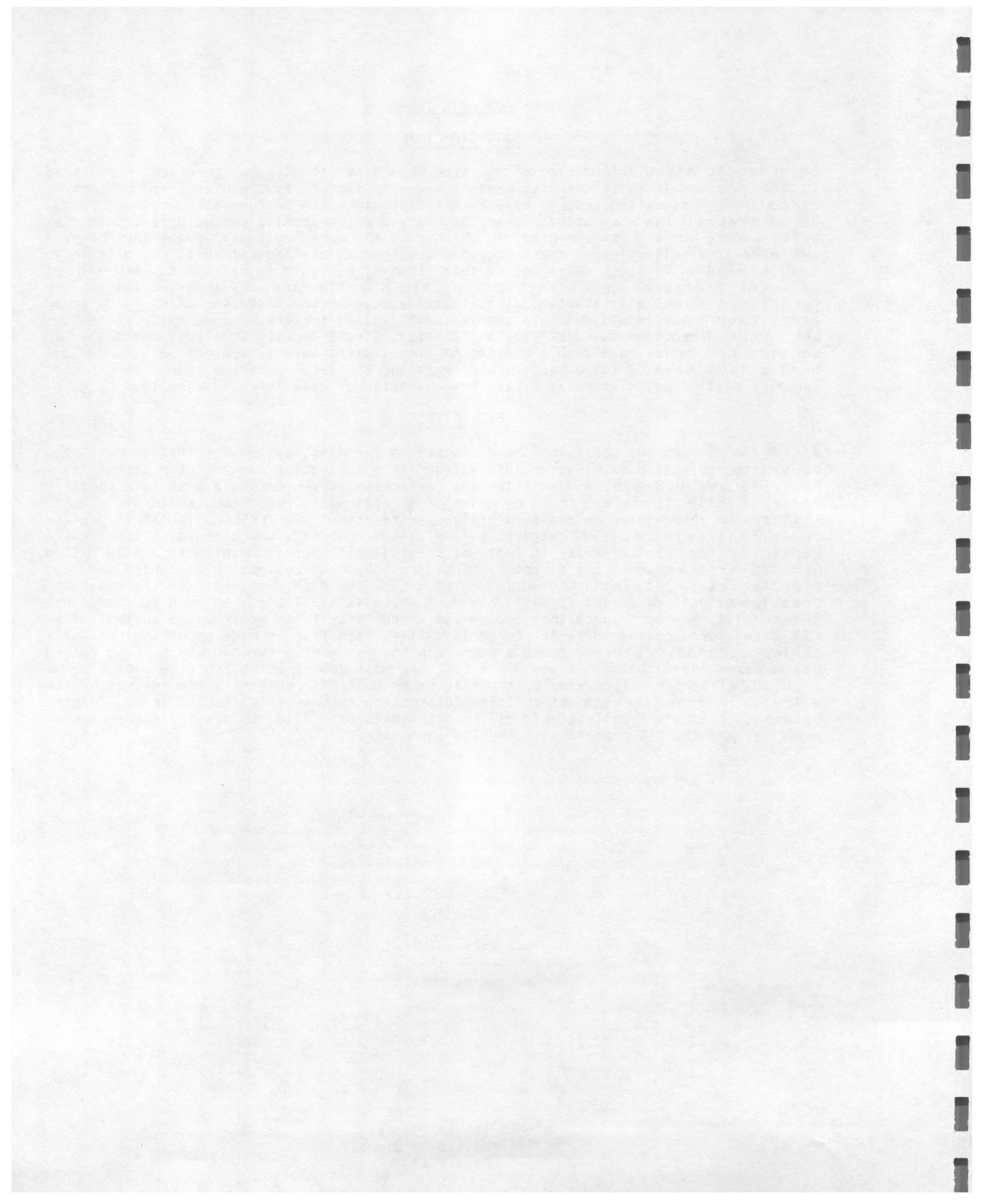




TABLE OF CONTENTS

FM - Overview .....	3
---------------------	---

DRIVE NUMBER USAGE:

	QUICK	DETAIL
In Display mode .....	4	14
In Kill mode .....	4	15
In Move mode .....	4	16
In the Remove Mode .....	5	17
USING PARTSPECS: .....	5	17
WILDCARD CHARACTERS: .....	6	21

PARAMETERS:

EXISTENCE PARAMETERS:

ABS .....	7
CK (Check NEW with Move or Remove) .....	7
N (New) .....	7

See individual Display, Kill, Move or Remove sections for detailed descriptions of ABS, CK, and NEW

DISPLAY PARAMETERS:

A (Allocation) .....	7	24
G (Gran) .....	7	24
K (1024 byte blocks) .....	8	25
O (Sort) .....	8	25
P (Printer) .....	8	26
Q (Query) .....	8	26

ATTRIBUTE PARAMETERS:

I (Invisible) .....	9	27
S (System) .....	9	27
V (Visible) .....	9	27
CM (Clear MOD) .....	9	27
SM (Set MOD) .....	9	28
FB (Use Flag Bit) .....	9	28
CFB (Clear Flag Bit) .....	9	28
SFB (Set Flag Bit) .....	10	28
PR (Use Protection Level) .....	10	29

DATE PARAMETERS:

D (Date) .....	10	29
SD (Set Date) .....	10	30
T (Today) .....	10	30

MODIFICATION PARAMETERS:

M (Modified) .....	11	31
U (Unmodified) .....	11	31

SIZE OF FILE PARAMETERS:

L (Limit) .....	11	31
Z (Size) .....	12	31

	<u>QUICK</u>	<u>DETAIL</u>
JCL OPERATION PARAMETERS:		
JCL .....	12	32
ABORT .....	12	32
FILE OUTPUT PARAMETERS:		
FI (File) .....	12	32
STR (Substitution string) .....	12	33
E (Enter character) .....	13	33
FINAL MESSAGE DISPLAY:		
In the Display mode .....		34
In the Move mode .....		34
In the Kill mode .....		34
In the Remove mode .....		34
ERROR HANDLING:		
Disk I/O errors .....		34
HELP SCREEN DISPLAY .....		36
CUSTOMER SERVICE AND LIMITED WARRANTY .....		37



## FM UTILITY PROGRAM

FM stands for File Manager. It is a utility program designed to facilitate specific manipulation of files. Four modes are available - Display, Kill, Move, and Remove, with the Display mode being the default. The basic syntax and available parameters for the command are:

```
=====
| FM partspec:d (parm,parm,...)
| FM partspec:d1 :d2 (parm,parm,...)
|
| FM partspec:d (KILL,parm,parm,...)
| FM partspec:d1 :d2 (KILL,parm,parm,...)
|
| FM partspec:d1 :d2 (MOVE,parm,parm,...)
| FM partspec:d1 :d2 :d3 (MOVE,parm,parm,...)
|
| FM partspec:d1 :d2 (REMOVE,parm,parm,...)
| FM partspec:d1 :d2 :d3 (REMOVE,parm,parm,...)
|
=====
```

The first three modes correspond with the commands DIR, PURGE, and the BACKUP by Files utility. The Remove mode is a combination of BACKUP and PURGE, with the files being moved from a source to a destination drive, and then being removed from the source drive. One of the main features of FM is that file operations can involve more than the usual number of drives. This provides comparison opportunities that can simplify the maintenance of diskette sets. The partspecing abilities of FM include three wildcard characters as well as the ability to specify a separate filename and extension for comparison purposes. The parameters include those previously available with BACKUP, such as modified, visibility status, file dating, etc. New parameters deal with recent dates, unmodified files, a way to set or clear mod flags without actually moving the files, and more. Due to the increase in popularity of large volume disk drives, several special parameters have been added to facilitate moving files from these larger drives onto smaller volume diskettes. Lastly, the speed of moving files has been increased by approximately 50% over the normal BACKUP by files, yet still includes a full read verify.

For reference purposes, the wildcard characters and the parameters are:

```
=====
| WILCARDING CHARACTERS:
|   $ Used as a masking character
|   * Used to indicate an Instring function
|   ! Used to indicate truncation of character checking
|
| A (Allocation), O (Sort), P (Printer), Q (Query)
| GRAN(G), K (1024 byte blocks)
| NEW (N), ABS, CK (Check)
| MOD (M), U (Unmodified), VIS (V), INV (I), SYS (S)
| DATE (D), TODAY (T)
| CM (Clear Mod), SM (Set Mod), SD (Set Mod date),
| CFB (Clear Flag Bit) SFB (Set Flag Bit) FB (Use Flag)
| PR (Use Prot)
| SIZE (Z), LIMIT (L)
| JCL, ABORT
| FILE (FI), STR (Substitution string) ENTER (E)
| KILL (KI), MOVE (MV), REMOVE (RMV)
|
=====
```

As can be seen from the layout of the first command block, FM has four modes (Display, Kill, Move and Remove), and can be considered to have three fields for any mode; the PARTSPEC, the DRIVE NUMBERS, and the PARAMETERS. Since the function of the drive number(s) is very important to FM, it will be explained first. The explanation of partspecs will follow, with the parameters being discussed last.

#### DRIVE NUMBERS:

The use of drive numbers depends on the mode used. In the Display and Kill modes, either one or two drive numbers can be used. In the Move or Remove modes, at least two drives must be specified, and three drives can be involved. When multiple drive numbers are used, the first will be always be referred to as the SOURCE DRIVE, and the second as the DESTINATION DRIVE. This will be true regardless of the mode involved. When using a three drive Move command the third drive will be referred to as the COMPARISON DRIVE. Although the following examples all use drives 0 and 1, any drive numbers may be used.

#### IN THE DISPLAY MODE:

```
FM :0  
FM :0 :1
```

These two commands show the use of drive numbers in the Display mode. The first command means "Show me all the files on drive 0 (the source drive) that match any partspec or parameters I have specified". The second command means almost same thing, except that using the second drive number says "Display the files on drive 0 based on a comparison of the files on drive 1". The important thing to remember is that the displayed files will always be files that are on drive 0, the source drive. Therefore, the Display mode can be used to "preview" the action of a Kill or Move command, and can assure that only the desired files will be acted upon.

#### IN THE KILL MODE:

```
FM :0 (KILL)  
FM :0 :1 (KILL)
```

These two commands show the use of drive numbers in the Kill mode. The first command means "Kill all the files on drive 0 (the source drive) that match any partspec or parameters I have specified". The second command means almost same thing, except that using the second drive number says "Kill the files on drive 0 based on a comparison of the files on drive 1". The important thing to remember is that the Killed files will always be files that are on drive 0, the source drive in this example. No file on drive 1 will be affected. To be assured that proper files will be killed, first use the identical command without the KILL parameter (i.e., the Display mode) to view the files that matched the specifications used.

#### IN THE MOVE MODE:

```
FM :0 :1 (MOVE)  
FM :0 :1 :2 (MOVE)
```

The Move mode is a little more complex as far as the use of drive numbers is concerned. The important thing to remember is that files will always be moved from the first drive to the second, regardless of the use of a third drive. When using a two drive Move command, the second drive, besides being the destination drive, also becomes the comparison drive. Thus the two drive command means "Move files that match the partspec and parameters from drive 0 to drive 1, based on a comparison of drive 1". The three drive command means "Move the files from drive 0 to drive 1 based on a comparison between drives 0 and 2". To be sure the proper files will be moved, use the command without the MOVE parameter to display the matching files. For the first example, the corresponding Display mode command would be FM :0 :1, as only two drives are involved.



To view the matches of the second Move example, the Display mode command FM :Ø :2 should be used, as drive 2 rather than drive 1 is the comparison drive.

### IMPORTANT

The NEW and ABS parameters, as discussed in a later section, are very important in MOVE and REMOVE commands. If in doubt concerning their use, study the detailed discussion section.

#### IN THE REMOVE MODE:

FM :Ø :1 (REMOVE)  
FM :Ø :1 :2 (REMOVE)

The Remove mode uses drive numbers in the same manner as the Move mode. That is, that files will always be moved from the first drive to the second, regardless of the use of a third drive. When using a two drive Remove command, the second drive, besides being the destination drive, also becomes the comparison drive. Thus the two drive command means "Move files that match the partspec and parameters from drive Ø to drive 1, based on a comparison of drive 1, and then kill the files on drive Ø". The three drive command means "Move the files from drive Ø to drive 1 based on a comparison between drives Ø and 2, and then kill the files on drive Ø". To be sure the proper files will be moved, use the command without the REMOVE parameter to display the matching files. For the first example, the corresponding Display mode command would be FM :Ø :1, as only two drives are involved. To view the matches of the second Move example, the Display mode command FM :Ø :2 should be used, as drive 2 rather than drive 1 is the comparison drive.

#### PARTSPECS:

Partspec stands for Partial File Specification. A file specification is defined as being a filename up to 8 characters long followed by an optional extension of a "/" followed by up to three characters. A partspec is considered to be any or all parts of a file specification. One important point to remember is that passwords are not necessary in any FM command, and should NEVER be used.

#### SPLITTING PARTSPECS:

The normal LDOS use of partspecs allows specifying a filename, an extension, or a filename/ext. The NOT symbol "-" can also be used to show an exclusion partspec such as -filename, -/ext, or -filename/ext. FM supports these standard uses of a partspec and adds to them! To accomplish this, FM allows a comma "," to be used as a separator between the filename and extension. Additionally, a filename may now include a trailing "/", meaning only files with no extensions. The following table lists the different partspec modes:

filename or -filename This type or partspec lets you specify either an inclusion filename or an exclusion filename. Any file whose filename matches the criteria will be a match regardless of the presence or absence of an extension.

filename/ or -filename/ This example is similar to the previous one, except that only files without extensions will be considered.

/ext or -/ext This type or partspec lets you specify an inclusion or exclusion extension. Since all files must have filenames, any file with an extension matching the criteria will be valid.

filename/ext or -filename/ext This example lets you specify an inclusion or exclusion filename as long as the files all match the extension criteria.

{-}filename,{-}/ext The power of having separate filenames and extensions makes itself felt by having the ability to specify an inclusion or exclusion filename, and a separate inclusion or exclusion extension. The braces "{}" mean that the NOTs are optional; the braces themselves should not be included on the command line. Following are the combinations using the separation of filename and extension:

filename,/ext -filename,/ext filename,-/ext -filename,-/ext

#### WILDCARD CHARACTERS IN PARTSPECS:

FM allows the use of three wildcard characters whenever a partspec is used. These are \$ (Mask character), \* (Instring character), and ! (Truncate character). More than one of these characters can be used at the same time, and in either the filename and/or extension fields.

\$ - The mask character is used to indicate that any character in a given position will be considered a match. For example, a partspec of \$\$A means "match all files that have an A as the third letter of the filename, regardless of the other characters in the filename. The \$ may also be interspersed in a filename, such as T\$\$60\$1. Using the \$ as a trailing character, such as TE\$\$\$, does NOT mean "only those files starting with TE and that contain 5 characters". The normal LDOS partspecing takes a partspec "TE" to mean "all files starting with TE, regardless of any other characters which may follow". To specify filenames of a certain length, see the ! character. The \$ character can be used in the extension part of a partspec as well as in the filename portion.

\* - The instring character is used to find a match of a specified string of characters anywhere in a filename or extension. The string to find may contain the \$ mask character. For example, a partspec of \*AT would find a match in files named PATCH, ATTACK, FORMAT, etc. A partspec such as \*B\$\$ would find files such as BASIC, OLDBUSI, etc. A partspec of /\*C would match all files that had a C anywhere in the extension. As filenames and extensions can be a maximum of 8 and 3 characters long, respectively, the characters following an \* are limited to 7 for a filename and 2 for an extension.

! - The truncate character is used to indicate files that have a filename or extension of a specified length. For example, a partspec of ALF! would match a file named ALF, but not ALFA1, ALFALFA, or AL. The \$ character can also be used with the !. A partspec such as TE\$\$\$! would find all files that start with TE and exactly five characters long. A partspec of /T\$! would match all files whose extensions start with a T and are exactly two characters long.

#### PARAMETERS:

The parameters of FM can be considered to be grouped into classes by function. Certain parameters deal with the attributes of files, some with dates, and others with size. To allow FM to be controlled by a JCL file, the JCL and ABORT parameters are included. The KILL, MOVE and REMOVE parameters switch FM out of its normal Display mode. One group of parameters deals with the type of display you will get from FM; sorted or unsorted, to the video or printer, and prompt or go nonstop.

#### FILE EXISTENCE PARAMETERS:

The NEW and ABS parameters can be used to override the default "Old comparison" feature of FM. NEW is valid in all four modes, while ABS is only valid for the Move and Remove modes. The CK (Check) parameter can be used along with NEW to determine if there is enough room on the destination files when moving files from one drive to another.



### NEW (N)

The NEW parameter can be included in any FM command that uses two or more drives; in one drive commands, NEW is meaningless. The normal default for any FM command regardless of the mode can be considered to be OLD. This means that only those files that exist on both of the drives will be considered. For example, the command "FM :0 :1" will show only those files that are on both drives 0 and 1. To see the files that are on drive 0 but not on drive 1, the command "FM :0 :1 (NEW)" can be used. The NEW parameter is valid in all four modes.

### CK (Check)

The purpose of the CK parameter is to check the amount of free space on the destination disk when the NEW parameter is used along with MOVE or REMOVE. If there is not enough free space to hold the new files, an appropriate error message will be shown, and the operation will abort.

### ABS

The ABS parameter can be used in the Move and Remove modes. The normal functioning of these modes is to only consider files that exist on both drives specified. If ABS is used, FM will act on the files whether or not they exist on the destination (or comparison) drive.

### DISPLAY PARAMETERS:

#### A (Allocation)

The A parameter is used to display certain information about a file besides the filename and extension. The format of the display will be:

```
|MM/DD/YY| nnnR  
+MM/DD/YY+ nnnR
```

The difference between these two examples is the use of | and + separators surrounding the date field. The date enclosed is the Mod date, and shows the date on which the file was created or last written to. If the | character is used, it means that the file's mod flag is not set. A + character indicates that the file has been modified (the FM parameters SM and CM may be used to adjust a file's Modified condition). The "nnnR" is the file's size in full disk records (256 byte blocks). The GRAN and K parameters may be used to switch from the number of records to the number of grans of a specific size, or to the number of 1024 byte blocks. In the Display mode, the normal screen or printer output will show the files in four across format. The A parameter switches to single line output. The Kill, Move, and Remove modes automatically show the mod date and file size.

#### GRAN=nn (G)

The GRAN parameter is used to display the size of a file in grans. Since LDOS supports many different disk types, the number of sectors per gran to use for the calculation may be entered by the user. If the GRAN parameter is used with no value, a 6 sector gran will be assumed. This will match the format of a 5" double density disk. The total grans of all matching files will be shown on the last line of the display.

### K (1024 byte blocks)

The K parameter is used to display the size of a file in K, or 1024 byte blocks. No "partial" K will be shown; any remainder will be rounded up to the next larger value. For example, 4.5 K will show as 5K. The total K of all matching files will be shown on the last line of the display.

### O=ON/OFF (Sort)

Functionally, the O parameter is used to turn OFF the alphabetic sort feature of FM. Normally the Display mode shows files sorted alphabetically, regardless of the order in which they are encountered when searching the disk. This is very useful for locating a particular file. On the other hand, the Kill, Move, and Remove modes act on the files in the order they are encountered. Thus, at times it may be desirable to display the files in their unsorted order before using one of those commands. If O is not specified when doing a Display mode FM command, it defaults to on.

### NOTE

Sorting the file specifications requires a certain amount of free memory. If there is not enough memory available, FM will not display the files in alphabetical order. This will be a very unusual occurrence, but can happen if free memory drops below a certain point.

### P (Printer)

Normally, the files acted on are shown on the video display. The P parameter provides a way to send the results to a printer. In the Display mode, the files are normally sent to the printer four across. If the A parameter is used, each file will be printed on a separate line. If the P parameter is used in the Kill, Move, or Remove mode, the file display will also be sent to the printer in single line format.

### Q=ON/OFF (Query)

The Q parameter is used in all four FM modes. With the Kill, Move and Remove modes, it lets the user be queried whether to act on the currently displayed file. For Kill and Remove, the default is ON (query before a file is killed); for Move the default is off (move files without asking). The default may be overridden by specifying the desired state of the Q parameter. In the Display mode, a Q=OFF parameter will keep the screen display from pausing as it normally does every 11 lines, and will display all files non-stop regardless. In the Kill, Move, and Remove modes, the following prompt will appear if Query is ON:

(Y/N/C) ?

Pressing <Y> tells FM to act on the file; pressing either <N> or <ENTER> means to bypass the file without action. Pressing <C> will act on the file and turn OFF the Query function. FM will continue nonstop from this point.

### FILE ATTRIBUTE PARAMETERS:

The next group of parameters lets the user specify groups of files based on their directory attribute type or modification status, as well as whether or not they exist on a destination or comparison drive. Two parameters, SM (Set Mod flag) and CM (Clear Mod flag) are provided to either set or clear the modification flags on specified files. Three parameters deal with the "Flag bit". This bit is also used by the PDS utility (a product of MISOSYS) to indicate its files. LDOS indicates files with this bit set by displaying an asterisk after their name in a DIRECTORY display. FB will find all files with the Flag bit set, SFB will set the Flag bit on matching files, and CFB will clear the Flag bit. The PR parameter will find all files matching a specified protection level, 0-7 (as described in the LDOS library command ATTRIB).



#### INV (I)

The INV parameter means act on only those files that are marked in the directory as being INVisible. It can be used in conjunction with the VIS and SYS parameters as required.

#### VIS (V)

The VIS parameter means act on only those files that are visible in a directory display. If neither INVisible nor SYStem are used as parameters, FM will use VIS as a default. To combine directory type parameters, use a command such as (V,I) meaning both visible and invisible, (V,S) meaning visible and system files, or (V,I,S) meaning all file types.

#### SYS (S)

The SYS parameter is used to specify LDOS system files. It can be used along with the VIS and INV parameters to specify file groups as needed. When using the S parameter in the Move or Remove mode, FM will correctly place LDOS system files in their required directory slots. If the SYS0/SYS file is moved, FM will also move the System Information Sectors as described in the LDOS documentation. This means that the state of any sysgened configuration, the state of the power-up date and time prompts, and the default drive configurations will be moved to the destination disk.

#### CM (Clear Mod flag)

The Display mode, using the CM parameter will clear the mod flag from any specified file. In the Move mode, using CM will always clear the mod flag on the source and destination drives. The Remove mode clears the mod flag on the destination drive. The Kill mode ignores this parameter.

#### SM (Set Mod flag)

In the Display mode, using the SM parameter will set the mod flag on every file that matches the specified partspec and/or parameters. In the Move mode, using SM will set the mod flag on both the source and the destination drive. In the Remove mode, the destination mod flag will be set. Since the source file will be killed, the SM parameter does not consider it.

#### FB (Use Flag Bit)

The "Flag" bit is a method of marking a group of files. The LDOS compatible PDS utility program also makes use of this directory marking feature. FM provides the ability to mark desired files in this manner, thereby allowing them to be Displayed, Moved, or Killed as a group. If a file has this bit set, it will appear in a normal LDOS DIR command with an asterisk after its name. The FB parameter can be used to find these files.

#### CFB (Clear Flag Bit)

This parameter provides a means to clear the user "Flag" bit discussed in the FB and SFB parameters. To see files that have this flag bit set, do a DIR command and observe those files that have an asterisk after the filespec.

### SFB (Set Flag Bit)

This parameter provides a means to set the user "Flag" bit for a file. It can be used as a means of marking a group of files to be manipulated in a common manner. To see files that have this flag bit set, do a DIR command and observe those files that have an asterisk after the filespec.

### PR (Use PROTection Level)

In the Display mode, this parameter limits the display to those files that have a matching Protection level. Normally, this protection is linked with the use of a password assigned to a file. The different modes will act on only those files that match the protection level. This protection level may be assigned with the LDOS library command ATTRIB.

### DATE SPECIFICATION PARAMETERS:

Three FM parameters deal with a file's date; two of them are used to select files based on a date, while the third can be used to change a file's date. The term "date" when used in reference to a file is the mod date - the date the file was created or last written to.

#### DATE (D)

The date parameter is used to specify a single date or a range of dates that will be used to determine if a file should be acted upon. Four forms of the parameter are allowed:

D="MM/DD/YY" Files only with the specified date.

D="M1/D1/Y1-M2/D2/Y2" Files with dates between the first and the second, inclusive.

D="-MM/DD/YY" Files with dates less than or equal to the specified date.

D="MM/DD/YY-" Files with dates greater than or equal to the specified date.

The D parameter may be used in all modes.

#### SD (Set Mod date)

The SD parameter is used to set the modification date of specified files. The date can be set either to the current system date or to a user specified date. The format for this parameter is:

SD This command will use the current system date.

SD="MM/DD/YY" This command will use the specified date string.

In the Move or Remove mode, using SD will set the dates on the destination drive files. The files on the source drive will retain their original dates.

#### TODAY (T)

One of the most common uses of the date parameter is to deal with files that have been recently modified. In recognition of this, the TODAY, or T, parameter has the ability to define a range of dates relative to the current setting of the LDOS system date. The allowable range is from seven days previous to the current date. The format of the T command is:



T This command means files with today's date.

T="n" where "n" is a number 1 to 7; 1 meaning yesterday and 7 meaning seven days ago.

T="n1-n2" where n1 is greater than n2. This command means "from n1 days ago to n2 days ago".

T="-n" This command means "from seven days ago to n days ago".

T="n-" This command means "from n days ago to today".

The T parameter can be used all modes.

#### FILE MODIFICATION PARAMETERS:

The file modification parameters are used to select either files that have been modified or files that have not been. With LDOS, a file is considered to be modified if it has been written to since it was last backed up.

##### MOD (M)

The MOD parameter can be used to specify only those files that have their "Mod" flags set. In normal LDOS operation, a file's mod flag will be set any time a file is written to or changed in any manner. The FM parameters SM (Set Mod) and CM (Clear Mod) can be used to adjust a file's Mod status.

##### U (Unmodified)

This parameter can be used to specify only those files that have not been written to or changed. As with the MOD parameter, the SM and CM parameters can be used to change a file's Mod status.

#### FILE SIZE PARAMETERS:

Two parameters, Limit and Size, can be used very effectively in systems that need to move data from large volume drives to smaller ones. While the FM utility does not provide a way to move a single large file onto a series of smaller disks, it does give the user an easy means to bypass these large files in Move or Remove commands, and then see which files were not moved.

##### LIMIT (LM)

The Limit, or LM, parameter is used to specify a maximum file size. The value for the Limit parameter will usually reflect the size of the destination disk to be used in the Move mode. For example, a 40 track, double density, single sided disk has 174K free after formatting. A 35 track, single density, single sided disk has 83K free after formatting. When moving files from a larger to a smaller disk, you can use the Limit parameter to have the FM utility ignore any files that will not fit on a single destination disk. For example:

FM :Ø :1 (MOVE,LM=174)

This command tells FM not to attempt to move files larger than 174K. Once the Move has finished, the Size parameter can be used to see any files that have not been moved.

### SIZE (Z)

The Size, or Z, parameter is used to view files that are larger than a specified length. This parameter can be used in conjunction with the Limit parameter when moving files from a large volume drive to multiple smaller disks. The format is:

SIZE=nnn

The "nnn" is the size in K (1024 byte blocks).

### JCL PARAMETERS:

The JCL (Job Control Language) parameters provide a means to run the FM utility under the control of the LDOS JCL processor. The use of these parameters is necessary due to the automatic error recovery feature of FM. During certain operations, FM will prompt the operator if a disk I/O error has occurred, asking whether to retry the operation, skip the file in question, or abort the operation entirely. This unexpected prompt could cause an active JCL to get "out of sync", and random results could follow. To avoid any problems, the following two parameters can be used.

#### JCL=Retries

This parameter tells FM that the current operation is being controlled by a JCL file, and also lets the operator specify the number of automatic retries to be attempted in case of an error. If JCL is used with no value, two retries will automatically be done. In either case, if the retries are not successful, the file will be skipped and the FM operation will continue. The one exception to this is the occurrence of a "Disk full" error, which will abort an FM command running under JCL control.

#### ABORT

The ABORT parameter can be used to override the automatic retry function that occurs when running under JCL control. If ABORT is specified, the occurrence of any error will abort the FM command, and the JCL processing.

### OUTPUT FILE PARAMETERS:

The output file parameters of FM are ENTER, FILE and STRING. They provide a means of writing any matching filespecs to an ASCII disk file. If desired, the filespecs can be embedded in a larger ASCII string, with the entire string being written to an output file. The main purpose of this feature is to create JCL files that can later be executed to perform a desired function on a group of files.

#### FILE="filename"

The FILE parameter is used to specify the name of the output file. If no extension is specified, /JCL will be the default. If the FILE parameter is used with no "filename", a file called FMOUT/JCL will be written. If an error condition is encountered when writing the output file, a single error message will be displayed but the FM command in progress will continue. The STRING parameter can be used to specify the characters that will be output to the file. If it is omitted, the only characters written to the output file will be the filespec, including the drive number of the source drive.

#### STRING="desired output string"

The STRING parameter can be used to specify the line sent to an output file. It allows for insertion of matching filespecs with and without drive number and password, and can embed <ENTER> characters in the string. Each string sent will automatically be terminated with a carriage return. Three special characters are used to accomplish this. They are:



& Will translate to FILENAME/EXT.PASSWORD:D \*\*  
| Will translate to FILENAME/EXT  
; Will translate to <ENTER> (Note that the <ENTER> translation can be assigned to some key other than the ";" with the ENTER parameter).

\*\* The PASSWORD mentioned in this section will only be used in the version for LDOS 5.1. Those using LDOS/TRSDOS 6.0 can disregard any reference to it. The PASSWORD referred to will be .EZTO, which is a substitute for the LDOS 5.1 master password. For example, consider the following FM command:

FM data:0 :1 (FI,STR="RENAME & /OLD")

If the files DATA1/ARS and DATA2/ARS matched the command, the resulting output file would have the lines:

RENAME DATA1/ARS.EZTO:0 /OLD  
RENAME DATA2/ARS.EZTO:0 /OLD

Once the FM command had completed, issuing a DO FMOUT/JCL would perform the desired renaming function.

#### E (Enter Character)

This parameter provides a means to designate a character that will be translated into a carriage return, or "Enter" character, in an output string. The default character value is a semicolon ";". If it is necessary to substitute some other character, the format E=value, or E="character" can be used. Specifying E=NO will cancel this feature.

## DRIVE NUMBERS

Since FM was designed to be a utility for file comparison, drive numbers play an important part in the final results of an FM command. For ease of explanation, this section will be broken up into four parts; the Display Mode, the Kill mode, the Move mode, and the Remove mode. In all modes, it is permissible to use more than one drive. Since the NEW and ABS parameters greatly affect the results of multiple drive commands, they will also be discussed in this section. The CK (Check before move) parameter will be detailed in the Move and Remove sections.

### IMPORTANT POINT

The FM utility normally operates with an implied "Old" parameter whenever two or more drive numbers are used in a command. This means that ONLY the files that exist on both drives will be considered as matching. The NEW parameter will override this feature in all four modes, and the ABS parameter may be used in the Move and Remove modes as explained in the following sections.

### IN THE DISPLAY MODE:

The normal use of a file display utility is to show the names of files on a particular drive. FM does support this standard feature. In addition, FM allows the display to be based on a comparison of one drive to another.

When using a single drive number, the FM display mode is similar to the normal DIR Library command. The files on the specified drive that meet any partspec criteria or parameters are displayed. The syntax of a display mode single drive command would be:

```
FM :d (parm,parm,...)
FM partspec:d
FM partspec:d (parm,parm,...)
```

As these general descriptions show, the display of files on a drive can be based on partial file specifications as well as other parameters. The important point to note: it is permissible to specify only one drive.

To use the comparison feature of FM, two drive numbers must be specified.

### IMPORTANT POINT

When using two drives on the command line, the resulting display will always consist of files from the FIRST DRIVE NUMBER specified.

The most basic use of two drive numbers in the display mode is to ascertain which files exist on both of the specified drives. For example:

```
FM :d1 :d2
```

As stated earlier, FM uses a default "Old" parameter. Thus, this command would display only the files on the first drive (d1) that were also on the second drive (d2). Partspecs and/or parameters could also be included in this type of command.

The NEW parameter will reverse this condition, allowing the display of files that exist on the first drive, but not on the second:

```
FM :d1 :d2 (NEW)
```



This command would show only those files that existed on the first drive, but were not found on the second. Again, partspecs and parameters could have been used.

As seen in both this example and the previous, the files displayed are always those that exist on the first drive number specified.

One of the most important uses of the Display mode is to preview the results of a Kill or Move command. Since those commands can greatly affect your files, the Display mode should be used to determine if the FM command to be used will produce the desired results. In the description of the Kill and Move parameters, use of the Display mode will be detailed.

#### IN THE KILL MODE:

The Kill mode of FM is used to remove files from a specified drive. Like the Display mode, either one or two drive numbers can be used. A single drive command will use only partspec information or parameters as criteria to determine which files will be killed. A two drive command will allow files on a second drive to be used for comparison purposes, but will still kill files on the first drive specified.

A single drive Kill command can have one of the following forms:

```
FM :d (KILL,param,param,...)
FM partspec:d (KILL)
FM partspec:d (KILL,param,param,...)
```

In these single drive examples, it should be obvious that the files to be killed are all on drive "d", and will be killed as long as they match the user entered partspec and parameters. When the Kill mode is used in this manner, it is very similar to the PURGE Library command, although it does provide added partspecing and parameter capabilities. The real power of the Kill mode can be seen when two drive numbers are used.

The syntax of two drive Kill command is:

```
FM :d1 :d2 (KILL,param,param,...)
FM partspec:d1 :d2 (KILL)
FM partspec:d1 :d2 (KILL,param,param,...)
```

In its basic form, this type of command means "Kill the files on drive d1, only if they match my partspec and parameter criteria, AND ONLY IF THEY EXIST ON DRIVE d2." Thus, the FM Kill mode can be easily used to remove duplicate files from disks. As stated earlier, the NEW parameter can be used to reverse the comparison, allowing FM to kill files only if they DO NOT exist on the comparison drive. For example, the command:

```
FM partspec:d1 :d2 (KILL,NEW,param,...)
```

would remove FROM DRIVE D1, those files that matched the partspec and parameters but DID NOT EXIST ON DRIVE D2. The important point to note is that whether or not the NEW parameter is used, the files that match the criteria will always be killed on the first drive specified! The second, or comparison, drive will never be altered.

When using a two drive Kill command, there is a method to preview the results of a Kill without actually killing the files. By issuing the command without using the KILL parameter, the matching files will merely be displayed on the screen. Then if everything is correct, you can re-issue the command with the KILL parameter included. Note: when using the Display mode, the files are normally shown sorted alphabetically. In the Kill mode, the files are shown in the order in which they are encountered on the disk. If you wish the Display mode to show the files in the order in which they will be killed, use the parameter (SORT=NO).

### IN THE MOVE MODE:

Move mode drive usage is slightly different from the Display and Kill modes in that it requires a minimum of two drive numbers, and will accept three. The NEW and ABS parameters will be discussed in this section, as they globally determine which files will be moved.

When using two drive numbers, Move is similar to the BACKUP utility using the (Old) parameter. The format of this type of Move command is:

```
FM :d1 :d2 (MOVE,param,param,...)
FM partspec:d1 :d2 (MOVE)
FM partspec:d1 :d2 (MOVE,param,param,...)
```

This type of command says "Move the files from drive d1 to drive d2, based on any other criteria I have specified, IF AND ONLY IF THE FILES ALREADY EXIST ON DRIVE D2." This is unlike the normal Backup-by-files which moves files regardless of their existence on the destination disk. As an example, let us assume that drive 1 contains a disk with files you wish to move, and drive 2 contains a newly formatted disk containing no files. If you were to issue a "FM :1 :2 (MOVE)" command, no files would be moved to drive 2, since there are no existing files on drive 2! There are two ways to circumvent this default "old" parameter. One way is by use of the ABS parameter; the other is by using the NEW parameter (which invokes the opposite restriction).

### The ABS parameter:

The ABS parameter is used to turn off the default "Old" parameter of the Move mode, and tells FM to move all matching files from the first drive to the second, regardless of their existence on the second drive. In this manner, FM will act like a normal backup by files command. The syntax of the command is:

```
FM :d1 :d2 (MOVE,ABS)
```

The use of a partspec and any additional parameters is optional. This command would tell FM to move all the files from drive "d1" to drive "d2", whether or not they already existed on drive "d2".

### The NEW parameter:

In a two drive Move command, the NEW parameter means "Move those files from drive :d1 to drive :d2, IF AND ONLY IF THEY DO NOT ALREADY EXIST ON DRIVE :d2". The syntax of the command is:

```
FM :d1 :d2 (MOVE,NEW)
```

This type of command has many practical uses, such as updating an archive disk with new files. Any files that already exist on the destination disk will not be touched.

### THREE DRIVE MOVE COMMANDS:

The three drive Move command allows the movement of files from one disk to another, based on the results of a comparison with a third drive. One of the most practical uses of this type of command is to move groups of files from a large drive to smaller one, based on comparison to another smaller drive. In this manner, the file extension or date can be readily used to create very specific archive or backup sets of disks.



When using the Move mode of FM with three drive numbers, it is important to remember the following:

The first drive number is always the SOURCE of the files to be moved.

The second drive number is always the DESTINATION of the files to be moved.

The third drive number is always the COMPARISON drive to determine the files to be moved.

The basic three drive Move command syntax is:

```
FM :d1 :d2 :d3 (MOVE,param,...)
FM partspec:d1 :d2 :d3 (MOVE)
FM partspec:d1 :d2 :d3 (MOVE,param,...)
```

In the three drive Move mode of FM, the NEW parameter is valid although the ABS parameter is not. The three drive Move command will function as follows:

NEW parameter not used - only those files that exist on both drive "d1" and "d3" will be moved from drive "d1" to drive "d2".

NEW parameter was specified - only those files that are on drive "d1" but NOT on drive "d3" will be moved from drive "d1" to drive "d2".

As can be seen from these examples, the third, or comparison drive, will never be changed.

#### IN THE REMOVE MODE:

The Remove mode is a "Move then Kill" command. The Remove mode command line syntax is the same as that of the Move mode with the exception of using the REMOVE, rather than the MOVE, parameter. The drive usage is identical.

In two drive Remove commands, files are moved from the first drive to the second, and then killed on the first drive. Partspecs and parameters can be used to identify the files to be acted upon.

In three drive Remove commands, files will be moved from the first drive to the second based on a comparison of the third drive, and then the files that were moved will be killed on the first drive.

#### The ABS parameter:

As with the Move mode, the ABS parameter is valid only with two drive commands, and means "move all files that match the partspec and/or parameters from the first drive to the second, and kill every file on the first drive after it has been moved."

#### The NEW parameter:

Again, this parameter means the same to Remove as it does to Move. However, any matching files will be killed on the first drive after they have been moved to the second.

#### FM PARTSPECING AND WILDCARDING

Besides the ability to use multiple drives for comparison purposes, FM provides another very powerful file inclusion/exclusion feature by its use of partspecing and wildcard characters. A partspec, or "partial file specification", means all or part of a file

specification. One of the main uses of partspecing is to select a group of files that have similar names or extensions, or both. For FM purposes, a standard filespec is defined as:

#### FILENAME/EXT

FILENAME is from 1 to 8 alphanumeric characters, the first of which must be alphabetic. All files must have a filename.

/EXT is an optional extension of 1 to 3 alphanumeric characters, the first of which must be alphabetic.

(Alphanumeric - consisting of the letters A-Z, and the numerals 0-9. No other characters may be used.)

Since the four FM modes (Display, Kill, Move, and Remove) are all very similar in their command structure, the examples in this section will be valid no matter what mode is used. The next section of the documentation will explain the differences between using single versus separate filenames and extensions.

#### PARTSPECING: Use of Filenames

The FM utility allows the use of either a filename (inclusion) or NOT filename (exclusion) to be used to specify a particular group of files. The format for this type of partspecing is:

FILENAME (Inclusion - means include all files that match this filename)

-FILENAME (Exclusion - means exclude all files that match this filename)

A filename used as a partspec can be from 1 to 8 characters long. One very important point to remember is that checking for a match will be done only on the number of characters specified in your filename partspec. For example, if an FM command of CO:Ø were used, all of the following files would match, assuming of course that they were on drive Ø:

COMP/CMD	CO	CONFIG/SYS	CONTINUE
----------	----	------------	----------

As shown in this example, neither the number of characters AFTER the partspec nor the file extension are normally taken into consideration when matching a "filename only" partspec (this can be changed, however, as explained later in the Wildcard and Separate Extension sections). The main purpose of this type of filename partspecing is to choose a group of files that all start with a common prefix; a very handy feature when examining a disk that contains files from more than one application. The same method of matching holds true when using a filename as an exclusion spec. Thus, if an FM command of -CO were used, all files would be shown EXCEPT those that started with the letters CO.

It is also possible to specify only files that have no extensions. To do so, use a filename partspec as follows:

FILENAME/ or -FILENAME/

By using a "/" character with no following extension, FM assumes that only those files with no extensions should be matched. Again, the filename may be either an inclusion or an exclusion spec.



IN SHORT: Using a FILENAME or -FILENAME as a partspec will find a match with any file that starts with the partspec, and does not care what extension, if any, the file has. The FILENAME/ partspec is used to find files with no extensions.

NOTES: Much greater selectivity is possible by including wildcard characters in the filename partspec, or by including a separate extension, as will be explained later.

#### PARTSPECING: Use of Filename/Ext

Using a FILENAME/EXT partspec in an FM command is very similar to using a filename only. The only difference is that the file's extension must match the /EXT specified. It is important to understand that both a filename and extension will be evaluated the same way; only the number of characters used in the FM command will be considered when searching for a match. Thus a command such as CO/A would find a match:

CO/A                      CO/ASM                      CONFIG/ART                      COMP/ASM

As this example shows, any characters in excess of those used in the command are disregarded, either in the filename or extension. Again, using the NOT character will exclude any files that match. Thus a command of -CO/A would show files except those that had a filename starting with CO, and that also had an extension starting with A.

SUMMATION: Using a FILENAME/EXT or -FILENAME/EXT as a partspec considers only as many characters as were specified in the filename and extension.

#### PARTSPECING: Use of /Ext

Just as files can be included or excluded according to their filenames, their extensions may also be used as the criterion. Again, only as many characters as are entered in the /ext partspec will be considered for matching purposes. Thus, using the FM command /C:Ø would match files with extensions such as:

/CMD    /CCC    /COM    /CIM    /CE    /C

The more specific the partspec used in the FM command, the more specific the matching will be. Using the extensions displayed from the previous example, a partspec of /CC would only match the /CCC extension.

It is also allowable to enter a NOT extension partspec. For example, using an FM command of -/CMD would display all files EXCEPT those that had an extension of /CMD. A command of -/C would exclude any files that had an extension that started with /C, etc.

IN SHORT: The /EXT partspec is similar to the FILENAME partspec in that it uses only the number of characters entered when searching for a match in an extension, and that the NOT character can be used. However, the filename will not have any effect when using an extension-only partspec.

#### PARTSPECING: Using separate Filename and Extension

As the previous sections have demonstrated, anytime a single filename/ext partspec is specified, both the filename and the extension must match when searching for inclusion or exclusion files. The FM utility does, however, provide a way to specify a separate filename and extension in the same command. Moreover, either the filename or the extension or both may be NOT specifications. Refer to the following table, taking particular note of the comma separating the filename and extension:

FILENAME,/EXT  
-FILENAME,/EXT  
FILENAME,-/EXT  
-FILENAME,-/EXT

Using separate filenames and extensions is very similar to the normal partspecing previously described as far as searching for matches is concerned, using the normal FM "two part" procedure. The filename is first checked for a match, and then the extension. By using a comma to separate the filename and extension, it is possible to use either part SEPARATELY as an inclusion or exclusion criteria, totally independent of the other part. For example, suppose you had a disk in drive 1 that contained the following files:

FM10/ASM	FM11/ASM	FM1/CMD	FM1/TXT
FM/CMD	FM2/TXT	FM12/ASM	

Looking at the file extensions, it appears that there are three types of files on this disk; /ASM assembler source files, /CMD object code files, and /TXT text documentation files. Also, the files all have filenames starting with the letters FM. To act on any one group of files, a command such as:

```
FM fm/txt:1
FM /txt:1
```

could be used to show all /TXT files. If it were desired to move only the /TXT files to another drive, a command such as:

```
FM fm/txt:1 :2 (MOVE)
FM /txt:1 :2 (MOVE)
```

could be used. Now consider the case where you want to move all of the files except the /TXT files. This could be done with the command:

```
FM -/txt:1 :2 (MOVE)
```

So far, the need for using the separate filename and extension feature has not been demonstrated. This is because of the small number of files that are the example disk. In normal practice, several groups of files are stored on the same disk. Consider the following revised display of a disk in drive 1:

FM10/ASM	FM11/ASM	FM1/CMD	FM1/TXT
CON/CMD	CON/TXT	TEST/ASM	TEST/TXT

In cases where multiple file groups are stored on the same disk, the separate filename and extension feature could be used to match specific groups of files:

```
FM fm,-/txt (Match all files starting with FM except those that have a /TXT extension)
```

```
FM -fm,/txt (Match all files with a /TXT extension except those that start with FM. In this case, CON/TXT and TEST/TXT would match)
```

```
FM -fm,-/txt (Match all files that do not start with FM and that do not have a /TXT extension. In this case, CON/CMD and TEST/ASM would match)
```

IN SHORT: The separate filename and extension are generally used when it is desired to match a group of files on a disk that contains many different file types.



### WILDCARDING:

Wildcard characters, or WCC, can be used in filename and extension partspecs to specify different groups of files. In FM, three WCC are allowed:

- \$ - The "mask" character
- \* - The "instring" character
- ! - The "truncate" character

The \$ WCC can be used in combination with the \* and the !. However, the \* and the ! are mutually exclusive.

### WILDCARD: The \$ character

The \$ mask character is used as its name implies; to mask, or disregard, character positions when searching for a match. Its function is identical to the normal LDOS \$ WCC. It is used primarily to search for a particular part of a filename or extension at a known offset. For example, consider the filespec:

TEST101/V21

This filespec would be matched by using any of the following partspecs in an FM command (These are just some of the possible partspecs):

\$\$ST1        \$\$\$101        \$\$\$101/V21    /\$21  
T\$ST\$01/\$\$1 TEST1/V\$1

As can be seen from these examples, the \$ character is used as a mask, or "don't care" character. No matter what character is in the filename or extension at that point, it will always be considered a match. This type of wildcarding is very useful for files that have common groups of characters at a known place in the filename or extension. It should also be kept in mind if creating files for an application. For example, suppose a home checkbook balancing program used files with the filenames consisting of the first three letters of the month followed by the two digit year. A data disk for this program might contain the following files, among others:

JAN83/CKB        MAR83/CKB        APR83/CKB        FEB83/CKB

Using a partspec of \$\$\$83/CKB, or even \$\$\$83, would match these files. In cases where the desired field to match (in this case, the "83") is not at the same offset in the filename or extension, the \* WCC can be used to find a match.

Note that in none of the previous examples was the \$ ever used as the last character in the filename or extension. Although this would be permissible, it would serve no purpose whatsoever. Remember that normal partspecing only searches forward for as many characters as were entered on the command line (i.e. "FM1" would match FM101, FM15, FM1TEST, etc.), thus, any trailing \$ WCC would be unnecessary. If a specific LENGTH of filename or extension needs to be specified, see the ! WCC explanation.

IN SHORT: The \$ character is used to mask off, or cause a "don't care" condition for characters at certain offsets in the filename or extension. It should not be used as a trailing character.

NOTE: The \* and ! WCCs can also be used with the \$, and provide certain other matching features.

### WILDCARDING: The \* character

The \* is the "instring" WCC (instring means "in the string"). It is used to match a filename or extension partspec no matter where that partspec is embedded in the target filespec. For example, consider the FM commands:

```
FM *ba:1
FM /*ba:1
```

The first command means "match all files on drive 1 that have the letters "BA" anywhere in the filename, and would match files such as BASIC/CMD, TBA/CMD, MOMBA, NEWBAK/DAT. The second example says "match any file with the characters "BA" anywhere in the extension, and would match files like TEST/BAS, XMPL/TBA, ALLIN/BAK.

The length of the partspecs is limited to the \* plus 7 characters for a filename, and the \* plus 2 characters for an extension.

The Not character "-" can also be used with the \*, in commands such as:

```
FM -*ba:1
FM -/*ba:1
```

These types of commands would exclude any files that had the characters "BA" anywhere in the filename or extension. As listed in the PARTSPECING section of the documentation, the following FM commands can also use the \* character:

```
FM -*filename/ext
FM -*filename,/*ext
FM *filename,-/*ext
FM -*filename,-/*ext
```

The \* WCC can also be used with the \$ WCC. This type of implementation is best described through example:

```
FM *$$te
FM /*$a
```

In the first example, the command states "match those files that have a 4 character string ending in TE anywhere in the filename". Files such as SEPTEMBR/82, MYTEST, etc. would match. The second example means "match any file with a two character string ending with A in the extension." Files such as TEST/BAS, XMPL/TBA, etc. would match. Practical uses of the \* and \$ combination will depend upon the files that are to be manipulated. When creating files (such as with a word processor, mailing list program, etc.), keep in mind this feature of FM and make file groups that are easily displayed, moved, or killed.

IN SHORT: The \* character is used to provide an instring, or sliding field type of comparison function for matching filenames or extensions. It can be used along with the \$ WCC.

### WILDCARDING: The ! character

The ! character is used to truncate, or cut off, the search for a match of a filename or extension partspec. Its primary use is to specify a particular length of partspec. It is best demonstrated through example; consider the following FM command using a truncated filename:

```
FM te!
```



This command would match files such as:

TE/CMD      TE      TE/BAK      TE/X3

but not the files:

TEST      TERRA/101      T/BAK

As the example shows, this type of partspec is used to match files that have an EXACT length of two characters, with those two characters being "TE". Any files that with filenames shorter or longer than two characters will not match. The same will be true if the partspec is an extension:

FM /CM!

This command would match files such as:

TEST/CM    MYCMD/CM    NOWAY/CM    A/CM

but not files like:

TE/CMD    OVLY/C    DATA/CMA    READ2/COM

As shown, only those files that have exactly two characters in the extension, and those two characters being "CM", will match.

The Not character "-" may also be used with the ! WCC to specify an exclusion partspec. Commands such as:

FM -te!  
FM -/te!  
FM -this,/c!

are all acceptable, as well as the other command forms described in the previous Partspeccing sections.

To specify all files that have a definite length of filename or extension, the \$ WCC can be used along with the ! WCC in the following manner:

FM \$\$! (All files with exactly two characters in the filename, regardless of any extension)

FM \$\$!/ (All files with exactly two characters in the filename, as long as they have no extension)

FM /\$\$! (All files with exactly two characters in the extension)

The Not character "-" can also be used to exclude matches when using the ! WCC, such as:

FM -TE! (Exclude any files that have TE as the only characters in the filename)

FM -/\$! (Exclude any files that have a single character extension)

FM -\$! (Exclude any files that have a single character filename)

IN SHORT: The ! character is used to specify an exact length of filename or extension when searching for a match. It can be used along with the \$ character.

## FM PARAMETER DESCRIPTIONS:

Besides the mode parameters discussed earlier, FM has many other parameters to help select a certain group of files, perform directory updating functions, or to let FM run under control of the LDOS JCL (Job Control Language).

### DISPLAY FORMAT PARAMETERS:

#### A (Allocation):

The A parameter is used with the Display mode (the other modes automatically turn this parameter on). The normal Display mode will show the files in alphabetical order listed across the screen, such as:

```
FILENAME/EXT  FILENAME/EXT  FILENAME/EXT  FILENAME/EXT
```

If the A parameter is specified, the files will be shown one per line, with other information following the filename and extension. The format would be:

```
FILENAME/EXT  |MM/DD/YY| 0000 R
-----
aaaaaaaaaaaa *bbbbbbb* cccc d
```

aaa This field will be the filename and extension.

bbb This field is the file's Mod date. This is the date of creation, or the last date the file was written to.

\* The asterisks represent the "mod status" characters. If the file has been modified (written to) since it was last backed up, then the characters in these two positions will be plus signs "+". Otherwise, the vertical bars "|" will be used as separators.

ccc This field is the total number of "units" in the file. If not changed by specifying the G or K parameter, this will be the total number of disk sectors assigned to the file.

d This field will show whether the normal "R" (number of records) mode was used, or if the G or K parameter was used. It will appear as the letter R, G, or K.

Normally, a Display mode command is done without using the A parameter. However, if it is necessary to see the additional information, the A parameter can be specified. If the P (send output to printer) parameter is specified along with the A parm, the printer output will also be in single line format, and will contain the date and unit information.

#### G (Gran) or G=:

The normal "unit" display of FM is in Records, meaning the number of disk sectors used by a file. The G, or Gran, parameter is used to specify the output as the number of grans assigned to a file. The main use of this parameter will be when dealing with disks of different sizes (3", 5", 8", hard) or different densities (single, double, hard disk format). Since a gran represents the minimum number of sectors that can be allocated to a file, consider the following table which shows the minmum number of sectors (DDEN=Double density, SDEN=Single density):



5" Floppy, SDEN	- Gran = 5 sectors, 1.2K
5" Floppy, DDEN	- Gran = 6 sectors, 1.5k
5" Hard,	- Gran = 16/32 sectors, 4/8K
8" Floppy, SDEN	- Gran = 8 sectors, 2K
8" Floppy, DDEN	- Gran = 10 sectors, 2.5K
8" Hard,	- Gran = 16/32 sectors, 4/8K

As seen in this table, there is a wide variance in the number of sectors, or records, that make up a granule. Thus, a file that contained only 1 record would take up 1 gran on a disk; however, the total disk space used would vary greatly.

The G parameter should be entered followed by a value representing the number of sectors per gran. Referring to the previous table, it can be seen that the value for a 5", SDEN Floppy would be 5, and for an 8" SDEN Floppy would be 8, etc. IF the G parm is used with no value, 6 will be assumed (matching a 5" DDEN Floppy).

When files are to be moved from one type of disk to another, the G parameter can be used to show the total number of grans that will be needed. Whenever the Display mode is used, the ending line always will show the total number of units taken up by the matching files. If the G parameter is used, this unit total will be the number of grans needed to store the files on a disk of the type having that gran size. By examining the final display of total grans needed, and comparing it with the number of free grans available on the disk, it can easily be determined if the desired files can be moved. The LDOS Library command FREE, when used to show the free space map of a single drive, will also show the free granules available on that drive.

#### K (1024 byte blocks):

Like the G parameter, K can be used to shift the FM output display from the normal Record mode to show the number of K, or 1024 byte blocks a file uses. The number of K will be based on the actual number of records in a file, not on the space allocated to it on the disk. No partial K will be shown, and any remainder will be rounded up to the next higher K. For example, a file with six records takes up 1.5K, but will be shown as 2K.

The main use of the K parameter is to help determine the actual space used by files in a system with many different drive types. Like the G parameter, this can help determine the actual amount of space needed when moving files between drives. The total number of K for all matching files will be shown in the final display line. If the A (Allocation) parameter is used, the K for each individual file will also be displayed.

#### O (Sort):

The O parameter is used to turn off the normal alphabetic file Display mode sort. To use it, issue a command such as:

FM :O (O=OFF)

Although the Display mode normally shows files sorted alphabetically, the other modes act on files as they are found while scanning the directory. To allow the files to be displayed in the same manner, the O parameter can be turned off. This may be desirable to preview the results of a Kill, Move, or Remove command, seeing the files in their actual order.

#### NOTE

Sorting the file specifications requires a certain amount of free memory. If there is not enough memory available, FM will not display the files in alphabetical order. This will be a very unusual occurrence, but can happen if free memory drops below a certain point.

### P (Printer)

The P parameter provides a way to send the results of an FM command to a printer. In the Display mode, the files are normally sent to the printer four across. Each entry sent will include the filename, the extension (if applicable), and the drive number of the source drive. If the A parameter is also used, each file will be printed on a separate line. The file's date, modification status, and size will also be printed. The size will normally be in total records, but can be changed with the G or K parameters. If the P parameter is used in the Kill, Move, or Remove modes, the file display will also be sent to the printer as if the A parameter was also used.

### Q=ON/OFF (Query)

The Q parameter affects the four FM modes differently. In the Display mode, Query is normally on. A Q=OFF parameter will keep the screen display from pausing as it normally does every 11 lines, and will display all files non-stop. If the JCL parameter was used on the command line, it will automatically set Q=OFF for Display mode commands.

With the Kill, Move and Remove modes, the Q parameter lets the user be queried whether to act on the currently displayed file. For Kill and Remove, the default is ON (query before a file is killed); for Move the default is off (move files without asking). The default may be overridden by specifying the desired state of the Q parameter on the command line. In the Kill, Move, and Remove modes, the following prompt will appear if Query is ON:

(Y/N/C) ?

Pressing <Y> tells FM to act on the file; pressing either <N> or <ENTER> means to bypass the file without action. pressing <C> will act on the file and turn OFF the Query function. FM will continue nonstop from this point.

The user is cautioned against using the Kill or Remove modes with the Q=OFF parameter until familiar with the way FM operates. Killing files automatically is an easy way to clean up a disk, but can be disastrous if the wrong files are involved.

### FILE ATTRIBUTE PARAMETERS:

The next group of parameters lets the user specify groups of files based on their directory attribute type or modification status, as well as by a user specified flag bit. Certain of these parameters use existing information about a file, while others allow a file's information be changed.

Two parameters, SM (Set Mod flag) and CM (Clear Mod flag) are provided to either set or clear the modification flags on specified files. Three parameters deal with the "Flag bit". This bit is also used by the PDS utility (a product of MISOSYS) to indicate PDS files. LDOS indicates files with this bit set by displaying an asterisk after their name in a DIRectory display. FB will find all files with the Flag bit set, SFB will set the Flag bit on matching files, and CFB will clear the Flag bit. By setting either the mod flag or the user flag bit, files ordinarily dissimilar in filename, extension, or other attributes may now be easily manipulated as a group.

The PR parameter will find all files matching a specified protection level, 0-7 (corresponding to the levels as described in the LDOS library command ATTRIB). Files may also be selected based on their visibility and system status.



### INV (I)

The INV, or I, parameter means act on only those files that are marked in the directory as being INVisible. These normally would be LDOS Utility files. To view the invisible files on a disk, issue an FM :d (I) command. One of the reasons for a user to make a file invisible is to prevent directory display from appearing cluttered. While using this parameter will find invisible files, it will not change them to visible files. That function can be performed with the regular LDOS Library command ATTRIB.

The I parameter can be used in conjunction with the VIS and SYS parameters as required to search for all three file types.

### VIS (V)

The VIS parameter means act on only those files that are visible in a directory display. If neither INVisible nor SYStem are used as parameters, FM will use VIS as a default. In normal practice, the only files that are not marked as visible will be the operating system files and utilities, or special user files. Since most file maintenance is done on visible user programs and data files, and VIS defaults to on, this parameter will normally not have to be specified.

If it is necessary to deal with all file types, such as when making a copy of a system disk, use a command such as (V,I) meaning both visible and invisible, (V,S) meaning visible and system files, or (V,I,S) meaning all file types.

### SYS (S)

The SYS parameter is used to specify LDOS system files. It can also be used along with the VIS and INV parameters to specify file groups as needed. Its main use is to allow the creation or updating of system disks.

When using the S parameter in the Move or Remove mode, FM will attempt to correctly place LDOS system files in their required directory slots. If that directory slot is in use by some other file, FM will abort with an error message. This should not happen with the 5.1 version operating system, but may occur with the 6.0 version. To avoid any problems, the S parameter should only be used to move system files to either freshly formatted disks, or to existing system disks.

If the SYS0/SYS file is moved, FM will also move the System Information Sectors as described in the operating system's technical documentation. This means that the state of any sysgened configuration, the state of the power-up date and time prompts, and the default drive configurations will be moved to the destination disk.

### CM (Clear Mod flag)

In the Display mode, the function of the CM parameter is very straight forward. It removes the modification flag from all matching files on the source drive.

The CM parameter cannot be used in the Kill mode. It would make no sense to clear a file's mod flag when it is about to be removed from the disk.

In the Move mode, using CM will always clear the mod flag on the source and destination drives. For this reason, the source drive cannot be write-protected, or the FM command will abort with an error.

In the Remove mode, using CM will clear the mod flag on the destination drive. As with Kill, nothing will be done to the source drive, because the matching files there will be removed from the disk.

### SM (Set Mod flag)

In the Display mode, using the SM parameter will set the mod flag on every file that matches the specified partspec and/or parameters on the source drive. One of the main reasons for setting mod flags is to be able to then use the MOD parameter to move these files from disk to disk. For example, suppose a disk contained several files to be moved, but the files did not share common characteristics such as a unique extension, filename, or mod date. You could use FM to set the mod flag on these files, and then use an FM :d :d (MOVE,MOD) command to transfer the files to another disk. This procedure could be repeated as many times as necessary, because the mod flags will not normally be cleared by the Move mode.

The SM parameter is not valid in the Kill mode, because any matching files will be removed from the disk.

In the Move mode, using SM will set the mod flag on both the source and the destination drive. For this reason, the source drive cannot be write-protected, or the FM command will abort with an error.

In the Remove mode, the destination mod flag will be set. Since the source file will be killed, the SM parameter does not consider it.

### FB (Use Flag Bit)

The "Flag" bit is a special mark in a file's directory entry. It can be set or cleared with the SFB and CFB parameters, respectively. Files that have their Flag bit set will appear in a normal operating system DIR Library display followed by an asterisk. The main use of the Flag bit is to allow a group of files to be marked, whether or not they have any other common filespec or attributes. This will facilitate moving or killing these files as a group, without the need to issue many separate commands.

### CFB (Clear Flag Bit)

This parameter provides a means to clear the user "Flag" bit discussed in the FB and SFB parameters. To see files that have this flag bit set, do a DIR Library command and observe those files that have an asterisk after the filespec.

In the Display mode, using the CFB parameter will clear the Flag bit on any matching file on the source drive. It, of course, is not valid in the Kill mode, as any matching files will end up removed from the disk.

In the Move mode, the Flag bit will be removed from both the source and destination disks. Be sure that the source disk is not write-protected, or the command will abort.

In the Remove mode, the Flag bit will be cleared on the destination disk.

### SFB (Set Flag Bit)

This parameter provides a means to set the user "Flag" bit for a file. It can be used as a means of marking a group of files to be manipulated in a common manner. To see files that have this flag bit set, do a DIR command and observe those files that have an asterisk after the filespec.

The section on the SM (Set Mod Flag) parameter gives a discussion of why you may want to use a special method of marking files.



### PR (Use PROTection Level)

This parameter provides a way to specify files by their assigned operating system protection level. These levels are different between the 5.1 and 6.0 versions, so check the DOS manual to determine what level names correspond to which protection level numbers. A file's protection level can be changed with the ATTRIB Library command.

In the Display mode, this parameter limits the display to those files that have a matching protection level. Normally, this protection is linked with the use of a password assigned to a file. The different modes will act on only those files that match the protection level.

The PR parameter is used with a value 0 to 7, with 0 being the least protection (FULL access, usually the level on user programs and data files), and 7 meaning the greatest protection (NO access, normally used only on operating system files). If, for example, a disk contained program files protected as Execute Only, these files could be accessed with the command:

```
FM :d (PR=6)
```

Thus by using the ATTRIB Library command to set common protection levels on files, they may be manipulated as a group by FM.

### DATE SPECIFICATION PARAMETERS:

Three FM parameters deal with a file's date; two of them (DATE and TODAY) are used to select files based on a date, while the third, (SD), can be used to change a file's date. The term "date" when used in reference to a file is the mod date - the date the file was created or last written to. To view the current mod date for any file, the FM Display mode or the normal DIR Library command can be used, providing the A parameter is used in either case. For example:

```
FM TEST:0 (a)
DIR TEST:0 (a)
```

Either of these two commands will show all files starting with TEST, and will include the file's mod date, as well as other information. For a further explanation, see the FM section on the A (Allocation) parameter.

### DATE (D)

The date parameter is used to specify a single date or a range of dates that will be used to determine inclusion of a file. Four forms of the parameter are allowed:

D="MM/DD/YY" Files only with the specified date.

D="M1/D1/Y1-M2/D2/Y2" Files with dates between the first and the second, inclusive.

D="-MM/DD/YY" Files with dates less than or equal to the specified date.

D="MM/DD/YY-" Files with dates greater than or equal to the specified date.

The D parameter may be used in all modes. For example, assume that the current date is 07/10/83. To find all files that have been written to in the last four days, one of the following commands could be used:

FM :d (D="07/07/83-07/10/83")  
FM :d (d="07/07/83-")

This command would result in a display of all visible files that had mod dates of July 7th through July 10th, the current date. This same date parameter could now be used move or kill selected files in the group.

#### SD (Set Mod date)

The SD parameter is use to set the modification date of specified files. The date can be set either to the current system date or to a user specified date. The format for this parameter is:

FM :d (SD) This command will use the current system date.

FM :d (SD="MM/DD/YY") This command will use the date string specified by MM/DD/YY. The restriction on the range of acceptable dates are the same as those for the operating system.

This parameter is not valid in the Kill mode.

In the Move mode, using SD will set the date on the destination drive files. The files on the source drive will retain their original dates.

In the Remove mode, the date will be set on the destination drive files. Nothing will be doen to the source drive, as the matching files there will be killed.

#### TODAY (T)

One of the most common uses of the date parameter is to deal with files that have been recently modified. In recognition of this, the TODAY, or T, parameter has the ability to define a range of dates relative to the current setting of the LDOS system date. It is merely a shorthand form of the DATE parameter. The allowable range is from seven days previous, to the current date. The format of the T command is:

T This command means files with today's date.

T="n" where "n" is a number 1 to 7; 1 meaning yesterday and 7 meaning seven days ago.

T="n1-n2" where n1 is greater than n2. This command means "from n1 days ago to n2 days ago".

T="-n" This command means "from seven days ago to n days ago".

T="n-" This command means "from n days ago to today".

The T parameter can be used in all modes. Again, remember that it defines files with dates matching the specified range, and is merely a convenient shortening of the DATE parameter.

#### FILE MODIFICATION PARAMTERS:

The file modification parameters are used to select either files that have been modified or files that have not been. Generally speaking, the mod flag indicates that the file was written to since it was last backed up with the operating system BACKUP Utility. To determine if a file is considered modified, use an FM Display mode command with the A parameter, or a DIR Library command. In either case, a Plus sign "+" will appear after every modified file. The FM parameters CM and SM also can be used to clear or set a file's mod flag.



### MOD (M)

The MOD parameter can be used to specify only those files that have their "Mod" flags set. In normal LDOS operation, a file's mod flag will be set any time a file is written to or changed in any manner. As explained previously, the SM parameter of FM can also be used to set the mod flag on selected files. Some of the reasons for using mod flags other than the way the operating system normally does are mentioned in the SM parameter section.

### U (Unmodified)

This parameter can be used to specify only those files that have not been written to or changed. It, in effect, performs the opposite function of the MOD parameter. As with the MOD parameter, the SM and CM parameters can be used to change a file's Mod status.

### FILE SIZE PARAMETERS:

Two parameters, Limit and Size, can be used very effectively in systems where it is necessary to move data from large volume drives to smaller ones. While the FM utility does not provide a way to move a single large file onto a series of smaller disks, it does give the user an easy means to bypass these large files in Move or Remove commands, and then see which files were not moved.

### LIMIT (LM)

The Limit, or LM, parameter is used to specify a maximum file size. The value for the Limit parameter will usually reflect the size of the destination disk to be used in the Move mode, and must be expressed in K (1024 byte blocks). For example, a 40 track, double density, single sided disk has 174K free after formatting. A 35 track, single density, single sided disk has approximately 83K free after formatting. Attempting to move a single file that was larger than the free space on the destination disk would cause an endless series of "Disk Full - Swap Disks" requests, and the Move would not proceed past that point. To circumvent this problem, you can use the Limit parameter to instruct the FM utility to ignore any files that will not fit on a single destination disk. For example:

```
FM :0 :1 (MOVE,LM=174)
```

This command tells FM not to attempt to move files larger than 174K, the maximum available space on a 40 track double density floppy. Once the Move has finished, the Size parameter can be used to display any files that were above the limit.

### SIZE (Z)

The Size, or Z, parameter is used to view files that are larger than a specified length. This parameter can be used in conjunction with the Limit parameter when moving files from a large volume drive to multiple smaller disks. The format for the parameter is:

```
SIZE=nnn
```

The "nnn" is the size in K (1024 byte blocks). For example, using a SIZE=174 will display the files that contained more than 696 record blocks. Like the limit parameter, SIZE is primarily useful when moving files from large volume disks to smaller ones.

### JCL PARAMETERS:

The JCL (Job Control Language) parameters provide a means to run the FM utility under the control of the LDOS JCL processor. The use of these parameters is necessary due to the automatic error recovery feature of FM. During certain operations, FM will prompt the operator if a disk I/O error has occurred, asking whether to retry the operation, skip the file in question, or abort the operation entirely. This unexpected prompt could cause an active JCL to get "out of sync", and random results could follow. To avoid any problems, the following two parameters can be used.

#### JCL=retries

This parameter tells FM that the current operation is being controlled by a JCL file, and/or lets the operator specify the number of automatic retries to be attempted in case of an error. Normally, a specific value is used, and can range from 1 to 10. The format of a command would be:

```
FM :0 :1 (MV,JCL=3)
```

If JCL is used with no value, two retries will automatically be done. In either case, if the retries are not successful, the file will be skipped and the FM operation will continue. The one exception to this is the occurrence of a "Disk full" error, which will abort an FM command running under JCL control.

Even when not running under JCL control, the JCL parameter can be used to specify a number of automatic retries, in case the operator will be away from the computer during the operation, and does not wish to be prompted on errors. This will assure that the operation will complete. The FM final display message will include an extra line if any files were skipped when running with the JCL parameter active.

#### ABORT

The ABORT parameter can be used to override the automatic retry function that occurs when running under JCL control. If ABORT is specified, the occurrence of any error will abort the FM command, and the JCL processing.

This parameter should be used when it is desirable to abort an operation no matter what type of error is encountered, such as when making copies of very critical data.

### OUTPUT FILE PARAMETERS:

The output file parameters of FM are ENTER, FILE and STRING. They provide a means of writing any matching filespecs to an ASCII disk file. If desired, the filespecs can be embedded in a larger ASCII string, with the entire string being written to an output file. The main purpose of this feature is to create JCL files that can later be executed to perform a desired function on a group of files.

#### FILE="filename"

The FILE, or FI, parameter provides a way to generate an optional ASCII file containing certain user selected information including filename and/or drive numbers during an FM command. The entire filename and extension can be entered on the command line, or the FILE parameter can be used by itself. The results will be as follow:

FI="FILENAME/EXT" would use the user specified filename and extension.

FI="FILENAME" would use the user's filename and add the /JCL default extension.

FI with no value will default to a file named FMOUT/JCL.



If an error condition is encountered when writing the output file, a single error message will be displayed but the FM command in progress will continue. If this happens, the output file should be considered fatally flawed, and should NOT be used until it is examined. Probably the best thing to do is to kill it, and try the FM command again.

STRING="desired output string"

The STRING parameter can be used to specify the line sent to an output file. It allows for insertion of matching filespecs with and without drive number and password, and can embed <ENTER> characters in the string. Each string sent will automatically be terminated with a carriage return. Three special characters are used to accomplish this. They are:

- & Will translate to FILENAME/EXT.PASSWORD:D \*\*
- | Will translate to FILENAME/EXT
- ; Will translate to <ENTER> (Note that the <ENTER> translation can be assigned to some key other than the ";" with the ENTER parameter).

\*\* The PASSWORD mentioned in this section will only be used in the version for LDOS 5.1. Those using LDOS/TRSDOS 6.0 can disregard any reference to it. The PASSWORD referred to will be EZTO, which is a substitute for the LDOS 5.1 master password.

The STR parameter used without any value (i.e., a command such as FM :0 (FI,STR)), will force the output string to consist of only the filespec, password, and drive number of the source drive. Otherwise, any characters included between the quotes will be sent to the output file, with the special characters being translated as described. For example, consider the following FM command:

```
FM data:0 :1 (FI,STR="RENAME & /OLD")
```

If the file DATA1/ARS matched the command, the resulting output file would have the line:

```
RENAME DATA1/ARS.EZTO:0 /OLD
```

Once the FM command had completed, issuing a DO FMOUT/JCL would perform the desired renaming function.

#### E (Enter Character)

The E parameter provides a means to designate a character that will be translated into a carriage return, or "Enter" character, in an output string. This assumes, of course, that the STR parameter was also used. The default character value is a semicolon ";". If it is necessary to substitute some other character, the format E=value, or E="character" can be used. Specifying E=NO will cancel this feature.

The main use of the semicolon or other ENTER character is to create a multi-line output string for later use. For example, if an FM command was used to find several files which were to be copied to another disk and then renamed, the following output string could be used:

```
FM TEST/NEW:0 (fi,str="copy & to |:1;rename |:1 /old")
```

Assuming that two matching files, TEST1/NEW and TEST2/NEW were found on drive 0, the resulting lines written to the output file would be:

```
COPY TEST1/NEW.EZTO:0 TO :1
RENAME TEST1/NEW:1 /OLD
COPY TEST2/NEW.EZTO:0 :1
RENAME TEST2/NEW:1 /OLD
```

Any number of embedded ENTER characters can be used, up to the point where the FM command will no longer fit on a single line.

#### FINAL MESSAGE DISPLAY:

When an FM command is finished, an appropriate message will be displayed. The exact message will depend on the mode (Display, Kill, Move, or Remove) that was selected, as well as the type of display requested (show Grans, K, or number of Records). In any case, the last segment of the sign-off message will be the total requested allocation of all files that matched the FM command line specifications and parameters. When it is desired to move files from one drive type to another, the total count from a display mode command can be used to determine how many destination disks will be required.

FM processing can be aborted at any time by pressing the <BREAK> key. If this is done, you will see the message:

Manual abort requested!

This lets you know that FM aborted before the end of its processing at your request. If FM finishes the request procedure, one of the following messages will be displayed, depending upon the mode selected.

Completed with AAA of BBB being matching files, CCC Total  
Completed with AAA of BBB being new files, CCC Total

Completed with AAA of BBB specified files moved, CCC Total

Completed with AAA of BBB specified files removed, CCC Total

In these messages, the "AAA" represents the number of matching files that were found or selected. The "BBB" represents the total files on the disk. (Note: the LDOS files BOOT/SYS and DIR/SYS are never considered in any FM command, and are NOT included in the "bbb" count). The "CCC Total" message will indicate the total Records, Grans, or K of the matching files. The first two messages will come at the end of a Display mode function. The others will appear after a Move, Remove, or Kill command.

If the JCL parameter was used, and I/O errors caused a file to be skipped, the following message will be displayed after the total line:

CAUTION - files bypassed in JCL mode!

#### ERROR CONDITIONS AND RETRY CAPABILITY:

When FM is performing a file display, a disk I/O error will abort the command. If this happens, the appropriate LDOS error message will be displayed, with an exit to the LDOS Ready prompt following. No final display message will be shown.

If an error occurs in the Move or Remove modes, FM will not automatically abort, but instead will give the operator a chance to retry the operation, skip over it, or abort as desired. This will be done regardless of a Query On/Off condition. Included in the error prompt message will be the drive the error occurred on, the operation FM was trying to perform at the time, the LDOS error number, and what options the user has.



Open error, Source \*\* nn, <R>etry, <S>kip, <BREAK> Abort  
Open error, Dest. \*\* nn, <R>etry, <S>kip, <BREAK> Abort  
Read error, Source \*\* nn, <R>etry, <S>kip, <BREAK> Abort  
Write error, Dest. \*\* nn, <R>etry, <S>kip, <BREAK> Abort

nn = The LDOS error number

If one of these error messages appears, there are three options available. Pressing <R> will retry the operation. The <S> option will skip over the current file and continue on with the operation. <BREAK> will abort the FM command entirely. To avoid aborting an FM command running under JCL control, the JCL parameter will assign a certain number of automatic retries. Once that number has been attempted, an automatic skip of the file will occur, and the operation will progress.

#### DISK FULL

One exception to the error trapping is a "Disk full" error during a Move or Remove command. If a disk full error is encountered, the following message prompt will be displayed:

Destination disk full replace it and press <ENTER>

If this message appears, remove the destination disk, replace it with the desired disk, and press <ENTER>. To abort the move, press <BREAK>. In either case, the last file displayed on the screen (the one that was caused the disk full error) will have been killed off the destination disk. If FM is being controlled by a JCL procedure, the occurrence of a disk full error will abort the JCL processing and display the appropriate error message.

## FM HELP SCREEN DISPLAY:

FM has a built in HELP! display function. If incorrect partspecs, drive numbers, or parameters are used, the offending command line will be shown, along with the Brief Help Screen listing. This Brief Help listing shows the available parameter abbreviations and wildcard characters only:

FM - Conditional File Exam/Kill/Move Utility - Ver. x.x.x

Command:

WILDCARD CHARACTERS \$=Mask, \*=Instring, !=Truncate

A, CK, P, Q, O, G, K, N, M, U, V, I, S  
D, T, SM, CM, SD, Z, LM, PR, SFB, CFB, FB  
JCL, ABORT, ABS, MV, KI, RMV  
FI, STR (&, |), E

The main function of this short Help display is to show the parameter abbreviations to help jog the memory. Note: the command FM (HELP,P) will send the full Help display to the printer rather than to the video. The display is:

FM ps:D (PARMS) or FM ps:D :D (PARMS)	for Display Mode
FM ps:D (PARMS) or FM ps:D :D (PARMS)	for Kill Mode
FM ps:D :D (PARMS) or FM ps:D :D :D (PARMS)	for Move/Remove

WILDCARD CHARACTERS: \$=Mask, \*=Instring, !=Truncate

DISPLAY: A=Allocation, P=Printer, O=Sort, G=Grans, K=1024 bytes  
I=Inv, S=SYS, V=Vis, M=Mod, U=Unmod, FB=Flag bit, PR=Prot level

SM=Set Mod, CM=Clear Mod, SD=Set Date  
SFB=Set Flag bit, CFB=Clear Flag Bit

DATE: D="mm/dd/yy-mm/dd/yy", T=Today, or T="Then-Now"

Q=Query before action, N=New files only, ABS=Disregard Old  
CK=Check Dest. space when Moving w/New

SIZE: LM=Limit to spec. K, Z=Show over spec. K

AUTO: JCL=Auto retries, ABORT=Quit on error

FILE OUTPUT: FI="Output File Name", STR="Output string"

STR sub. chars: &=FILENAME/EXT.PSWD:D, |=FILENAME/EXT  
E=Character Value for embedded Carriage Return



Logical Systems Incorporated  
The Help Systems

T A B L E   O F   C O N T E N T S

Introduction.....	page 1
Warranty.....	page 1
Overview.....	page 2
HELP/CMD.....	page 3
HELPRESx/CMD.....	page 6
HELPGEN/CMD.....	page 9
Appendix A.....	page 12
Appendix B.....	page 13
Appendix C.....	page 17





## The LDOS HELP Systems

### Introduction

This documentation covers all three HELP packages. Certain pages may not pertain to the individual package which you have purchased.

For a listing of the programs which should be contained on each package, refer to Appendix A of this manual.

The documentation additionally covers all three packages on two different operating systems, LDOS 5.1.3 and LDOS 6.1. Therefore, certain paragraphs will be different for each version. When this occurs, it will be prefaced by the appropriate version number. Please do not make the mistake of assuming that the sections which do not pertain to your specific version are relevant.

The reason for their inclusion is so that you may, at no additional expense, see exactly what the other packages are like.

### WARRANTY

All products sold by Logical Systems Incorporated, hereinafter referred to as LSI, grant the user certain customer support privileges. This support shall be limited to the privilege of having the master diskette updated as often as desired for the current update fee. This is limited to updates within the current Series of the program. LSI will also provide a lifetime warranty on the physical diskette media of the original serialized master diskette. If the diskette media physically fails to retain the original program, replacement media will be provided at no charge. This does not include media that has been damaged in shipment from the user to LSI, or media that has been damaged by the user or their equipment. To receive this support, the user MUST fill out and return a specific registration card pertaining to the product, within 30 days of purchase. Should a user find a valid error in the program and clearly define it in writing to LSI, every effort will be made to correct the error. All support shall apply only to registered owners.

Logical Systems Incorporated and its associates assume no liability whatsoever, with regard to the reliability and/or fitness of their products. All data entrusted to these programs and the computer that it is operating on are the sole responsibility of the user. Under no circumstances will LSI or its associates be held liable for the loss of TIME, DATA, PROGRAMS or for any consequential damages incurred by the user.

This manual, as well as the accompanying programs and data files, are Copyrighted © by Logical Systems, Incorporated, all rights RESERVED. Reproduction, by any means, and distribution is hereby forbidden except by written consent.

For additional information, please contact:

Logical Systems Incorporated  
P.O. Box 23956  
8970 N. 55th Street  
Milwaukee, Wisconsin 53223  
(414) 355-5454

## HELP SYSTEM

The Help System Utility is designed to convert any type of textual information into a readily accessed file which can then be displayed to the video screen. It is designed to be implemented either as a stand alone application, or from within a calling program. The Help system is comprised of the following Modules:

1. HELP/CMD This program is the stand alone Help module which is invoked from the LDOS Ready prompt, or from within languages or programs that allow system commands to be executed.
2. HELPRES/CMD This program resides the HELP System in high memory so that it may be accessed by applications which do not normally provide access to LDOS system functions.
3. HELPGEN/CMD This program converts any suitably structured ASCII file into a data file capable of being acted upon by the HELP System.
4. LDOS/HLP This data file contains help for most of the LDOS library commands and utilities.
5. LBASIC/HLP This data file contains help for LBASIC commands and functions.
6. Z80A/HLP This data file contains help for Zilog Z-80 mnemonics which start with the letters "A" through "L". Included information is: flags affected, opcode, timings and definitions of operations.
7. Z80M/HLP This data file contains the rest of the Zilog Z-80 mnemonics starting with the letters "M" through "X".
8. TECH1/HLP This is a reproduction of most of the information contained in the Technical section of the LDOS owner's manual. It contains information up to the System Entry Points section.
9. TECH2/HLP This is the rest of the information contained in the Technical Section of the LDOS owner's manual.

The following conventions are used in this manual:

- <> Encloses literal keyboard characters. <ENTER> is used to signify that the Enter key should be pressed. <BREAK> indicates depression of the Break key.
- fs Refers to a full LDOS standard file specification (filespec).
- ds Refers to an LDOS drive specification (drivespec).
- fn Refers to that part of a filespec which precedes the slash character (filename).
- kw A keyword used to find a specific area of HELP.
- p An optional parameter.

After purchasing the HELP system:

First, make two BACKUPS of the Master Diskette enclosed with the system. These diskettes should be used to generate future backups with the Master held as an ultimate reserve.

For the sake of "elbow room", it may be necessary to copy the files desired to various diskettes. The following chart gives the approximate length of the programs and data files: (The second number is for equivalent 6.1 files.)

File	Help File Length	Source File Length
LBASIC/HLP	24,871 [22,626]	30,208 [29,184] bytes
LDOS/HLP	47,851 [37,963]	57,856 [49,152] bytes
TECH1/HLP	54,498 [40,044]	68,864 [48,640] bytes
TECH2/HLP	50,016 [45,908]	60,672 [55,552] bytes
Z80A/HLP	46,431 [46,842]	62,464 [62,976] bytes
Z80M/HLP	28,834 [29,020]	40,704 [40,960] bytes



The actual HELP system display modules occupy the following space

HELP/CMD	4,044 [4,837] bytes
HELPRES1/CMD	3,569 [3,592] bytes
HELPRES2/CMD	4,409 [4,386] bytes
HELPGEN/CMD	3,674 [3,565] bytes

Distribution of user created files generated by HELPGEN/CMD are subject to the whim of the user, provided Logical Systems, Inc. is acknowledged. All files provided on the Master diskette are copyrighted including both HELP display modules. Those persons wishing to implement the HELP System in any distribution of their own should contact LSI at the address provided, concerning fees and contractual obligations.

Copyright © 1983 by Logical Systems, Inc.  
P.O. Box 23956  
8970 N. 55th Street  
Milwaukee, Wisconsin 53225  
(414) 355-5454

### HELP/CMD

This will probably be the most common use of the Help System. To obtain HELP from LDOS Ready, type the following:

HELP fn kw (p,p) <ENTER>

Where fn is the database filename to be searched, kw is the keyword desired in that file, and p represents any optional parameters.

For example, at LDOS Ready type:

HELP LDOS LIB <ENTER>

This will now open the file called LDOS/HLP, and display the information filed under the keyword "LIB". The video display will remain until any character generating key is depressed. If there is more information about the keyword than would fit on one screen, pressing a key will cause more information to display and, if necessary, pause repeatedly until the information is exhausted. At that time, the video display is restored and control will be returned to LDOS.

To inspect all the keywords contained within a file, type:

HELP fn <ENTER>

This will list all of the keywords within the named file. If the previous example had been "HELP LDOS <ENTER>", a list of the available keywords would have been displayed. Once again, the display will pause if necessary. After each screenfull of keywords, the prompt "<ENTER>, <BREAK>, or type keyword?" will appear. At this time, if <ENTER> is pressed, the next screenfull of keywords will be displayed. If <BREAK> is pressed, HELP will abort, and control will return to LDOS. A keyword may be entered, and the information relating to that keyword will be displayed.

If the specified keyword was not in the called file, the list of all keywords would display again, to indicate what was available within that file.

The entire command sequence can be entered in either upper or lower case.

To list all of the help files presently available on the system, merely type:

HELP <ENTER>

This will search all drives on line for files ending in the /HLP extension, and list them to the video display. For example,

Help Categories presently on line are:

Z80A/HLP:1	Z80M/HLP:1	LDOS/HLP:1	LBASIC/HLP:1
TECH2/HLP:3	TECH1/HLP:3		

Press ENTER to exit or enter category

The function of the HELP command may be altered by specifying one or more of the following optional parameters:

- P This parameter sends the output to the \*PR device (usually a printer) as well as the video. While using this option, the display will not pause if filled. Since all characters are being sent to the \*PR device, no pause is required.
- V This parameter causes the video restoration feature to be cancelled. If not specified, the screen will be returned to the same condition as it was when HELP was invoked, less the help command itself.
- B [5.1 Only] This parameter causes the blink feature to be cancelled. Various characters can be made to flash in the video display by specifying them as blinking characters during creation of the data file. However, if HELP were invoked while in a communications mode, a continuous stream of characters would be sent from the host machine to the terminal. The B parameter alleviates this difficulty.
- R [6.1 Only] This parameter causes the reverse video option to be cancelled. Various phrases can be displayed in a reverse video mode if so specified in the creation of the data file. However, certain terminals utilize the characters involved and unpredictable results can occur while in the communications mode. The R parameter alleviates this difficulty.
- S This parameter causes the Search mode to be entered. Typing:

HELP LDOS D (S)

would cause a listing of all keywords starting with "D" to be displayed rather than the entire list. The potential match should be the left most characters of a keyword. By specifying "DI", all keywords starting with "DI" would be displayed.



Besides use at LDOS Ready, the same command sequence may be employed within LBASIC by utilizing the CMD"exp" function. A user application could be written to invoke help as an operator choice from a menu or command line. For example, CMD"HELP LDOS FILESPEC" might be invoked by the application program if it detected an invalid filespec entry by the operator. HELP requires about 5 K of free memory to function. All system memory guides are followed, and the HELP system will abort if sufficient memory is not available.

Another example of a call from LBASIC might be: CMD"HELP LBASIC LSET" which will function (memory permitting) as described above, and would return control to LBASIC.

HELP/CMD also allows a "global" scan for any on-line keyword. If the keyword "MEMORY" was known, but the file is unknown (or to save typing in the filename), then enter an asterisk (\*) followed by the keyword in the help command line. For example,

#### HELP \*MEMORY

would find the first occurrence of the keyword "MEMORY" in any /HLP file. The top of the screen displays the category being scanned while a global search is in progress. If the key is found, the text displays normally. At the end of the text, the prompt "Press <BREAK> to exit or <ENTER> to continue global scan" appears. Pressing <ENTER> will look for the same keyword in another file until all /HLP files have been examined. Upon completion of the scan, or if no match is found, the normal prompt for category selection will appear. Continue as desired by pressing <BREAK> or <ENTER> to return to LDOS Ready or by typing in a category name to obtain the directory for that file.

## HELPRESx/CMD

HELPRES1/CMD or HELPRES2/CMD are similar in execution to HELP/CMD, but are filters that reside in high memory. This is so that applications programs which do not provide system access for LDOS commands MAY be able to call help from within. The difference between programs where HELPRESx/CMD will work and will not depends entirely on the calling program. In order to work, the target program must:

- A. Respect certain LDOS (and good programming) practices by respecting high memory modules. This means that almost any uncomplicated BASIC program would work because the code will not allow memory usage above the HIGH\$ limit.
- B. Must not use internal drivers for keyboard or video that cause the existing drivers to be ignored or unpredictable results can occur.
- C. Must allow sufficient memory space for HELPRESx/CMD to function.
- D. As a rule of thumb, generally, any program which allows use of the MINIDOS filter will work with HELPRES.

Unfortunately, LSI does not have the resources to make this determination on the thousands of programs available. It is simply not possible to deduce what logic each programmer utilized. Therefore, it is left up to the user to determine HELP's compatibility with existing software.

Fortunately, no harm can come of experimentation-- provided it is not done on the only existing copy of anything.

The difference between HELPRES1/CMD and HELPRES2/CMD is that HELPRES1 is able to display exactly one file and HELPRES2 can display from one through fifteen files simultaneously. Why are both included? HELPRES1/CMD needs 1740 [1856] bytes minimum while HELPRES2 requires 2025 [2127] bytes minimum.

To reside HELP in memory, the LDOS \*KI driver MUST be active (LDOS 6.1 always has this active). If it is, type the following:

### HELPRESx/CMD (p,p)

at the LDOS Ready prompt, where x is either 1 or 2.

### Parameter Functions:

- V     The V parameter will turn on or off the video restoration option. When off, screen restoration becomes the responsibility of the interrupted program. Programs such as Scripsit, or EDAS, can redraw the video display quite readily. Others, such as BASIC programs, were probably not written to refresh the screen while operating. Besides inherent program factors, keep in mind that the video refresh option requires an additional block of memory which equals the maximum number of characters contained on the video. For 16x64 screens this would be 1024 bytes, and for 80x24 screens this would be 1920 bytes. Once the V option is selected one way or the other it cannot be changed unless a reset is performed on the \*KI device.

### IMPORTANT NOTE

THE DEFAULT FOR HELP/CMD HAS THE VIDEO RESTORATION DEFAULTED TO "ON". IN THE INTEREST OF SAVING MEMORY, HOWEVER, THE HELPRES/CMD MODULE ASSUMES THE VIDEO RESTORATION TO BE "OFF". The V parameter will REVERSE the default condition!



B [5.1 Only] The "B" parameter will specify a maximum number of characters to blink per screen. Specifying "B=10" would, therefore, allow no more than ten blinking characters per screen. In the interest of memory management, be advised that each blinking character reserved, occupies 3 bytes of memory. A "B=10" will occupy 30 more bytes of high memory than having no "B=" in the command line. A maximum of 255 blinking characters may be specified by the B parameter. The default option is B=0, so that if no blink option is desired the user need not enter the parameter at all.

R [6.1 Only] Cancels reverse video just as it does for HELP/CMD

FILE HELPRES1/CMD requires a parameter of "FILE=". Unlike HELP/CMD, the filter may only look through 1 file at a time. To specify the LDOS/HLP file type, HELPRES1 (FILE="LDOS"). To change files after the filter is loaded, simply type another line at LDOS Ready with a different FILE= specified. The HELP filter will automatically substitute another file in the same memory location. The "FILE=" parameter must be specified for the resident module to work. A prompt will be issued demanding a file name if none is supplied or if an illegal filespec was entered. Entering a non-existent file name will abort the load. The FILE parameter may be abbreviated by its first letter "F".

FILE HELPRES2/CMD requires a parameter of "FILE=x", where x is a number from 1 through 15. HELPRES2 will then ask for each filename. It is also possible to use this parameter exactly as is done for HELPRES1. In this case, only one file is allowed. The abbreviation is "F".

DISABLE This parameter is used to cancel the filter. If either HELPRES is the last filter appended to the \*KI device, it will disengage and cease functioning. Furthermore, if it is the last high memory module in place, it will release the occupied memory. Because of this feature, HELPRES should be last. "DISABLE" may be abbreviated with "D".

To access the HELP System with HELPRES/CMD active, generate a <CLEAR><SHIFT><H> from the keyboard. This is accomplished by depressing the <CLEAR> key, then the <SHIFT> key, and finally the <H> key.

In HELPRES1 :

The video will prompt with "filespec Help: <BREAK>,<ENTER> or type keyword". Enter the keyword sequence desired, and the screen will display the information. If a non-existent keyword was requested, a list of available keywords will be displayed. Pressing <ENTER> will either display the file directory of keywords or continue one in progress. <BREAK> is pressed to return to the calling program. The video display will not be restored unless "V" was specified at the command line during start up.

In HELPRES2 :

The video will prompt with a "filespec Help: <CLR>,<BRK>,<ENT> or type keyword". Enter the keyword sequence desired, and the screen will display the information. If a non-existent keyword was requested, a list of available keywords will be displayed. Pressing <ENTER> will either display the file directory of keywords or continue one in progress. <SHIFT><CLEAR> is pressed to swap the active file with a dormant file. This will cause the "Select New Category" prompt to appear. If the resident files are forgotten, press <ENTER> and the list of files specified at start up will display and the "Select" prompt repeated. Type a category name or press <SHIFT><CLEAR> to return to the normal prompt. <BREAK> is pressed to return to the calling program from either keyword or category select. The video display will not be restored unless "V" was specified at the command line during start up.

Some problems may occur with either resident help system if certain guidelines are not followed.

The resident "helps" work by opening files on the disks. In the interest of shortening code, files are not opened and closed in high memory. Instead, a pointer is kept to the known position on a specific drive of a given file. Therefore, if a diskette containing an opened help data file (/HLP) is removed, it follows that the resident module is now dealing with erroneous information and unpredictable results will surely occur. REMEMBER to DISABLE either HELPRES program before exchanging or removing diskettes.

Because the resident helps are rather large, every effort has been made to make them convenient. Normally, a high memory module can neither be disengaged from the modified device without a RESET of the device, nor can the HIGH\$ pointer be re-routed to release the occupied memory without a BOOT. If either HELPRES is used with the "D" parameter then both can be achieved if this filter was the LAST high memory module loaded and/or the LAST device modifier for the Keyboard (\*KI).

Placing another filter below HELPRES will circumvent memory release because the lower module would then be unprotected in memory. If the latter condition occurs, HELPRES might unchain itself from the keyboard but still be resident in memory because of the "trap" set by the user.

If at that point, the SAME HELPRES is re-activated, the "trapped" code will re-activate rather than grabbing another swath of memory. If the opposite module is loaded, it will NOT use the same space. To utilize this re-activate feature, proceed as if swapping a file.

If a \*KI modifier is loaded after either HELPRES, then it is, of course, impossible to either unchain or release memory.

If it is desired to use another file instead of the file in memory, file swapping occurs as follows. Proceed as if the HELPRES is being used for the first time. It will detect its presence in memory and replace the file used now with the previously used file. However, ALL PARAMETERS except for the single replacement filename will remain as they were from the original installation. Note that HELPRES1 (F="newname") will replace the old file with the current file, but that B or R, and V will not be affected.

HELPRES2 works almost the same way. Only the "active" file (the one currently pointed to which is on the prompt line) can be replaced. To replace a "dormant" file, activate it with <SHIFT><CLEAR> from within the filter and then replace it. Again, other installation conditions (as well as all dormant files) are not affected.

For example, suppose that HELPRES2 is initialized with the files LDOS and LBASIC. When the program is entered the prompt says "LDOS Help" etc. To replace the LDOS file with Z80A type, HELPRES2 (F="Z80A"), at LDOS Ready. Now the filter is set for Z80A and LBASIC. To replace LBASIC, enter the program and switch LBASIC to an active state by using the <SHIFT><CLEAR> sequence. Return to LDOS Ready and type the proper command sequence. If large numbers of files are to be replaced, it may be more efficient to disable the resident module and re-initialize it rather than continuously rotating files and returning to LDOS Ready.

The runtime length of each module is approximately:

#### 5.1 Versions

HELPRES1	1740 bytes + 3 bytes/blink allowance + 1024 bytes with video restoration
HELPRES2	2025 bytes + blink + video + 41 bytes/file > 1

#### 6.1 Versions

HELPRES1	1856 bytes + 1920 bytes with video restoration
HELPRES2	2127 bytes + video + 41 bytes/file > 1



## HELPGEN/CMD

This module is a text processor which turns a file created under some type of text editor (WordStar, LED etc) into a data file which can be used by the HELP System. The file to be processed must be an ASCII file. Some text editors or word processors automatically save in ASCII while others use a non-standard, variable, compressed data storage format. For the latter reason, decoding for HELP is only done on the characters having an ASCII value of less than 128. The maximum destination file allowed is 65,535 bytes.

If "HELPGEN P" is specified at LDOS Ready, an alphabetical list of keywords found in the text file will be sent to the line printer.

The rules concerning creation of a source file to be processed by HELPGEN are as follows:

1. The first character of the file will be taken to be the first character of the first keyword. If this is a control character such as a carriage return or any other character with an ASCII value of less than 32, the processor will abort. If it is any non-alphanumeric character it will become part of the keyword. Be certain that the first character of the file is supposed to be the first character of the first keyword.
2. The "keyword" will continue until a carriage return <ENTER> character is encountered. NO other character will terminate a keyword. Thus, if the first sequence of characters were "SYNTAX<ENTER>", the keyword would be "SYNTAX". If the sequence were "LBASIC SYNTAX<ENTER>", the keyword would be "LBASIC SYNTAX". The keywords are used to access the file information as in "HELP LDOS SYNTAX" or "HELP LBASIC LBASIC SYNTAX" in the case of the two examples. The "keyword" should be chosen with this in mind. Alphabetic characters in the keyword will be converted to upper case in the index (HELP LDOS <ENTER> display). A keyword phrase may not exceed video line width less 14 characters. One further point on keyword length. Since the file directory prints a total list of keys for a file, long keys make the video display look sloppy.
3. Next, type in text as normal. This will become the information which displays when HELP fn kw is invoked. No line should exceed video width. For the most part, type the text exactly as it is desired to appear. A carriage return is not necessary after each line of text, however, remember that in that case, word location may not be the same as in the text editor.
4. [OPTIONAL 5.1 only] In the case of the LDOS/HLP file, blinking characters are used to signify abbreviations of parameters. Blinking characters provide emphasis for whatever reason desired. To make a character blink, type an ASCII value (127) immediately in front of the character desired. An ASCII 127 can be produced with the LDOS KI driver active by generating a <CLEAR><SHIFT><ENTER>. On Model I, the character is a block of dots, while in Model III it is a plus mark over an underline character. When output to the display, this character will be DROPPED from the display and the rest of the line will be moved one position to the left. Keep this in mind if column positioning is desired. Examine the proper use of this feature by observing it in the SAMPLE/TXT file. Many characters may be flashed in any single screen. Keep in mind that too many blinking characters can be aggravating to read. Also, if the text file is to be used in the filter mode, remember that a flashing character requires 3 bytes of memory.

4. [OPTIONAL 6.1 Only] Reverse Video may be used to provide emphasis. In the case of the LDOS/HLP file, reverse characters are used to signify that part of a parameter which can be used as an abbreviation. Reverse video for one or more characters is specified by typing an ASCII 127 character immediately preceding the first character to reverse, and another ASCII 127 immediately after the last character to reverse. An ASCII 127 is generated by pressing <CLEAR><SHIFT><ENTER>. The character appears as a plus mark on top of an underline.
5. End text entry for any given keyword's text by typing an ASCII 12. The next group of characters will then be another keyword. End this key with a carriage return and proceed to type more text. If there is no more information to follow, then the ASCII 12 should be the last character of the text file.
6. [OPTIONAL] If it is desired to have more than one key access the same text ("KEY STROKE MULTIPLY" and "KSM", for example) then type an ASCII value (12) character immediately after the carriage return of the previous keyword. The next sequence of characters will also become a keyword. This process may be used indefinitely. This would allow thousands of keys to access the same information. This is useful to provide for abbreviated keys (SYN and SYNTAX, EXT and EXTENSIONS, etc.), so that HELP LDOS SYN and HELP LDOS SYNTAX would have the same result. (Note that the LDOS/HLP file is done this way.). ASCII 12 is also sometimes called a forced end of page or top-of-form, or form-feed. Refer to the example in the SAMPLE/TXT file.
7. Repeat steps one through six until the source file is complete. The HELPGEN program compresses out spaces in the text and sorts the keywords automatically so no particular attention to the order of keywords and text need be observed. Because of space compression, source files longer than 65,535 characters can be processed. Average storage saved is about 20%.
8. Save the text file using an ASCII format. Be sure to specify ASCII, if it is an option on the text editor utilized.
9. Exit the text editor.

At this point, the text file is ready, and now must be processed for use by the HELP Display system, be it HELP or HELPRESx.

- A. At LDOS Ready, type HELPGEN (or HELPGEN P).
- B. HELPGEN will prompt for the source filespec. This is the ASCII file which was generated in step 8.
- C. HELPGEN will prompt for the HELP filename. Type in up to eight alphanumeric characters with the first character being alphabetic. The file will be ASSIGNED an extension of /HLP. Keep in mind that the name will constantly be used as part of the command line, and that the shorter it is, the easier it will be to use.
- D. If the file exists, a prompt will appear asking permission to overwrite the file. If the response is negative, the system will prompt for the filename again.
- E. Pressing <BREAK> in response to any prompt will abort the sequence.



SAMPLE/TXT is provided on the HELP Generator package to be an example of a proper source text. The user may wish to process this text as is, and then modify it to become familiar with the process. The source texts for the other HELP packages are also available should the user desire to alter or add to them.

Listed below are the error messages which may occur during processing. Any LDOS system errors incurred are explained in the LDOS manual.

**Improper Source Filename**

A source file which did not meet LDOS filespec standards was entered. A re-prompt for another filespec will occur.

**Duplicate Keys Encountered**

Two identical keywords were found. No recovery-- system aborts after displaying the duplicate key. Edit the text file and reprocess.

**Keyword exceeds maximum allowable length**

Be sure that a carriage return was used within the allowable maximum key length (video width less 14 characters). No recovery-- system aborts. Edit the text file and reprocess.

**Memory overlap has occurred due to too many keys**

The keyword list is too long to fit into available memory. No recovery-- system aborts. In borderline cases, releasing more high memory may help. It may be necessary to divide the source file.

**Same Source and destination file**

No recovery-- system aborts. Use a different destination filespec.

**Null Key Encountered - Prior Key was xxx**

A keyword contained no characters, which makes it rather tough to find. The Previous key (if any) is displayed to assist in the location of the offender. Normally, this is just a stray carriage return. Edit the source file and reprocess.

**Destination File Exceeds 65,535 characters**

No recovery - system aborts. Divide the source and process into two destination files. 64K is the maximum number allowed in the destination file. The source file may be any length because spaces are compressed out.

## Appendix A - - Files Included

LDOS HELP - - {Cat #'s L-30-060 (5.1), L-30-061 (6.1)}

LDOS/HLP  
LBASIC/HLP  
HELP/CMD  
HELPRES1/CMD  
HELPRES2/CMD

Technical HELP\* - - {Cat #'s L-30-080 (5.1), L-30-081 (6.1)}

Z80A/HLP  
Z80M/HLP  
TECH1/HLP  
TECH2/HLP  
HELP/CMD  
HELPRES1/CMD  
HELPRES2/CMD

\*REQUIRES at least two double density drives.

HELP Generator - - {Cat #'s L-30-070 (5.1), L-30-071 (6.1)}

HELPGEN/CMD  
HELP/CMD  
HELPRES1/CMD  
HELPRES2/CMD  
SAMPLE/TXT

HELP text source\* - - {Cat #'s L-31-010 (5.1), L-31-020 (6.1)}

Z80A/TXT, Z80B/TXT, Z80C/TXT  
Z80D/TXT, Z80E/TXT  
TECH1/TXT, TECH2/TXT, TECH3/TXT, TECH4/TXT, TECH5A/TXT  
TECH5B/TXT, TECH6/TXT, TECH7/TXT, TECH8/TXT  
LDOS1/TXT, LDOS2/TXT, LDOS3/TXT  
LBASIC1H/TXT, LBASIC2H/TXT

Z80A, B, and C are APPENDED to form Z80A and Z80D and E form Z80M. TECH1 is concatenated from TECHs 1, 2, 3, 4, and 5A. TECH2 is comprised of 5B, 6, 7, and 8. LDOS is derived by appending LDOS1, 2, and 3. LBASIC comes from LBASIC1H and 2H.

\*REQUIRES at least two double density drives.

NOTE : On LDOS HELP 6.1 the file BASIC/HLP is substituted for LBASIC/HLP and BASIC1/TXT, BASIC2/TXT are substituted for LBASIC1H/TXT and LBASIC2H/TXT respectively, in the L-31-020 package.



## Appendix B - - Keywords in the 5.1 Data Files

Help System Display Mode Version 5.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : LBASIC

&H	&O	CLOSE	CMD
CMD"*"	CMD"A"	CMD"B"	CMD"D"
CMD"E"	CMD"I"	CMD"L"	CMD"LDOS"
CMD"N"	CMD"O"	CMD"P"	CMD"R"
CMD"S"	CMD"T"	CMD"X"	CVD
CVI	CVS	DEF FN	DEFUSR
EOF	FIELD	GET	INPUT
INSTR	ERRORS	KILL	LBASIC ENTRY
LINEINPUT	LINEINPUT#	LOAD	LOC
LOF	LSET	MERGE	MID\$=
MKD\$	MKI\$	MKS\$	NPARMS
OPEN	PRINT#	PRINT# USING	PUT
RESTORE	RSET	RUN	SAVE
SET EOF	TIME\$	USR	XPARMS

Press <BREAK>, <ENTER> or type keyword

Help System Display Mode Version 5.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : LDOS

APPEND	ATTRIB	AUTO	BACKUP
BOOT	BUILD	CLOCK	CONV
COPY	COPY23B/BAS	CREATE	DATE
DEBUG	DEBUG D	DEBUG DISK	DEBUG E
DEBUG EXTENDED	DEVICE	DEVICES	DIR
DO	DUMP	EXT	EXTENSIONS
FILES	FILTER	FORMAT	FREE
HITAPE	JL	JOBLOG	KILL
KEY STROKE MULTIPLY	LCOMM	KI/DVR	LINK
KSM	LOAD	LIB	MEMORY
LIST	MINIDOS/FLT	LOG	PATCH
MINIDOS	PR/FLT	PASSWORDS	RDUBL
PDUBL	REPAIR	PURGE	ROUTE
RENAME	RUN	RESET	SPOOL
RS232X/DVR	SYNTAX	SET	SYS BASIC2
SYN	SYS BREAK	SYS ALIVE	SYS DATE
SYS BLINK	SYS FAST	SYS BSTEP	SYS SLOW
SYS DRIVE	SYS SYSGEN	SYS GRAPHIC	SYS TIME
SYS SVC	SYS UPDATE	SYS SYSRES	TIME
SYS TYPE	TWOSIDE	SYSTEM	
TRACE		VERIFY	

Press <BREAK>, <ENTER> or type keyword

## Appendix B - - Keywords in the 5.1 Data Files

Help System Display Mode Version 5.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : Z80A

ADC A,S	ADC HL,SS	ADD A,(HL)	ADD A,(IX+D)
ADD A,(IY+D)	ADD A,N	ADD A,R	ADD HL,SS
ADD IX,RR	ADD IY,RR	AND	AND TABLE
BIT B,(HL)	BIT B,(IX+D)	BIT B,(IY+D)	BIT B,R
CALL	CALL C,P	CCF	CP
CPD	CPDR	CPI	CPIR
CPL	DAA	DEC IX	DEC IY
DEC M	DEC RR	DI	DJNZ
EI	EX (SP),HL	EX (SP),IX	EX (SP),IY
EX AF,AF'	EX DE,HL	EXX	FLAG CODES
FLAGS	HALT	IM 0	IM 1
IM 2	IN A,(N)	IN R,(C)	INC (HL)
INC (IX+D)	INC (IY+D)	INC IX	INC IY
INC R	INC RR	IND	INDR
INI	INIR	JP	JP (HL)
JP (IX)	JP (IY)	JP C,P	JR
JR C,E	LD (BC),A	LD (DE),A	LD (HL),N
LD (HL),R	LD (IX+D),N	LD (IX+D),R	LD (IY+D),N
LD (IY+D),R	LD (NN),A	LD (NN),DD	LD (NN),HL
LD (NN),IX	LD (NN),IY	LD A,(BC)	LD A,(DE)
LD A,(NN)	LD A,I	LD A,R	LD DD,(NN)
LD DD,NN	LD HL,(NN)	LD I,A	LD IX,(NN)
LD IX,NN	LD IY,(NN)	LD IY,NN	LD R,(HL)
LD R,(IX+D)	LD R,(IY+D)	LD R,A	LD R,N
LD R,R'	LD SP,HL	LD SP,IX	LD SP,IY
LDD	LDDR	LDI	LDIR

Press <BREAK>,<ENTER> or type keyword



## Appendix B - - Keywords in the 5.1 Data Files

Help System Display Mode Version 5.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : Z80M

NEG	NOP	OR S	OR TABLE
OTDR	OTIR	OUT (C),R	OUT (N),A
OUTD	OUTI	POP	POP IX
POP IY	PUSH	PUSH IX	PUSH IY
RES B,S	RET	RET C	RETI
RETN	RL S	RLA	RLC (HL)
RLC (IX+D)	RLC (IY+D)	RLC R	RLCA
RLD	RR S	RRA	RRC (HL)
RRC (IX+D)	RRC (IY+D)	RRC R	RRCA
RRD	RST	SBC A,S	SBC HL,SS
SCF	SET B,S	SLA	SRA
SRL	SUB S	UNSIGNED COMPARISONS	
XOR	XOR TABLE		

Press <BREAK>,<ENTER> or type keyword

Help System Display Mode Version 1.0  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : TECH1

CMD FORMAT	DCB	DCB QR	DCB RAM AREAS
DCB+00	DCB+01,02	DCB+03,05	
DCB+06,07 (MOD1)		DCB+06,07 (MOD3)	
DCT	DCT QR	DCT+00	DCT+01,02
DCT+03	DCT+04	DCT+05	DCT+06
DCT+07	DCT+08	DCT+09	
DEVICE CONTROL BLOCK		DIR QR	DIR+00
DIR+02	DIR+03	DIR+04	DIR+05,12
DIR+13,15	DIR+16,17	DIR+18,19	DIR+20,21
DIR+22,23	DIR+24,25	DIR+26,27	DIR+28,29
DIR+30	DIR+31	DIREC	
DIRECTORY RECORDS		DISK COMMAND FILE FORMAT	
DISK I/O TABLE	DRIVE CODE TABLE		EQUATE1/EQU
EQUATE3/EQU	EXTENDED DIRECTORY RECORDS		FCB
FCB QR	FCB+00	FCB+01	FCB+02
FCB+03,04	FCB+05	FCB+06	FCB+07
FCB+08	FCB+09	FCB+10,11	FCB+12,13
FCB+14,15	FCB+16,19	FCB+20,23	FCB+24,27
FCB+28,31	FILE CONTROL BLOCK		
FILTERS & DRIVERS		FXDE	GAT
GAT QR	GAT+00,5F	GAT+60,BF	GAT+C0,CA
GAT+CB	GAT+CC	GAT+CD	GAT+CE,CF
GAT+D0,D7	GAT+D8,DF	GAT+E0,FF	
GRANULE ALLOCATION TABLE		HASH INDEX TABLE	
HIT	LOAD MODULE FORMAT		MOD1 MEMORY MAP
MOD3 MEMORY MAP		TAPE FILE OBJECT CODE FORMAT	

Press <BREAK>,<ENTER> or type keyword

## Appendix B - - Keywords in the 5.1 Data Files

Help System Display Mode Version 5.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : TECH2

@ABORT	@ADTSK	@CKDRV	@CKEOF
@CLOSE	@CMD	@CMNDI	@CTL
@DATE	@DEBUG	@DIV	@DODIR
@DSP	@DSPLY	@ERROR	@EXIT
@FEXT	@FNAME	@FSPEC	@GET
@INIT	@KBD	@KEY	@KEYIN
@KILL	@KLTsk	@LOAD	@LOC
@LOF	@LOGGER	@LOGOT	@MSG
@MULT	@OPEN	@PARAM	@PAUSE
@PEOF	@POSN	@PRINT	@PRT
@PUT	@RAMDIR	@READ	@REW
@RMTsk	@RPTsk	@RREAD	@RUN
@RWRT	@SKIP	@TIME	@VER
@WEOF	@WHERE	@WRITE	
BYTE I/O PRIMITIVES		CFCB\$	DATE\$
DAY\$	DCT\$	DCTBYT	DFLAG\$
DIRCYL	DIRRD	DIRWR	
DISK FILE HANDLER ROUTINES		DISK FILE HANDLERS	
DISK I/O PRIMITIVE NAMES		DISK I/O PRIMITIVES	
DIVEA	DODCB\$	DOSV\$	ERROR 00
ERROR 01	ERROR 02	ERROR 03	ERROR 04
ERROR 05	ERROR 06	ERROR 07	ERROR 08
ERROR 09	ERROR 10	ERROR 11	ERROR 12
ERROR 13	ERROR 14	ERROR 15	ERROR 16
ERROR 17	ERROR 18	ERROR 19	ERROR 20
ERROR 21	ERROR 22	ERROR 23	ERROR 24
ERROR 25	ERROR 26	ERROR 27	ERROR 28
ERROR 29	ERROR 30	ERROR 31	ERROR 32
ERROR 33	ERROR 34	ERROR 35	ERROR 36
ERROR 37	ERROR 38	ERROR 39	ERROR 40
ERROR DICTIONARY		EXDBG\$	
FILE CONTROL ROUTINES		GETDCT	HIGH\$
I/O CONTROL BLOCKS		INBUF\$	
INTERRUPT PROCESSOR TASK VECTOR		STORAGE	INTIM\$
INTVC\$	JDCB\$	JFCB\$	JLDCB\$
JRET\$	KEYBOARD I/O ROUTINES	KISV\$	KFLAG\$
KIDCB\$	KIJCL\$	MULTA	LDRV\$
MATH ROUTINES	MFLAG\$	PRDCB\$	OSVER\$
OVRLY\$	PDRV\$	PRSV\$	
PROGRAM ENTRY CONDITIONS			RDSECT
RDSSEC	ROM CONTROL ROUTINES		RSECT
S1DCB\$	S2DCB\$	S3DCB\$	S4DCB\$
S5DCB\$	SBUF\$	SEEK	SELECT
SFCB\$	SFLAG\$	SIDCB\$	SODCB\$
SPECIAL OVERLAY ROUTINES		SPECIAL PURPOSE	DISK ROUTINES
SUPERVISORY CALLS		SVC	SYSTEM BUFFERS
SYSTEM CONTROL INFORMATION		SYSTEM CONTROL ROUTINES	
SYSTEM ENTRY POINTS		SYSTEM FLAGS	
TASK CONTROL VECTORS		TCB\$	
TIME & DATE ROUTINES		TIME\$	TIMER\$
UNKNOWN ERROR CODE		VERSEC	
VIDEO & PRINTER I/O ROUTINES		WRPROT	WRSECT
WRTK			

Press <BREAK>, <ENTER> or type keyword



## Appendix C - - Keywords in the 6.1 Data Files

Help System Display Mode Version 6.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : BASIC

ABS	ABSOLUTE VALUE	ALTER LINES	ARCTANGENT
ARGUMENT	ARRAYS	ASC	ASCII VALUE
ASSEMBLY SUBROUTINE		ATN	AUTO
AUTOMATIC LINE NUMBERING		AVAILABLE MEMORY	
BRANCH TO SUBROUTINE		CALL	CANCEL ARRAY
CDBL	CHAIN	CHARACTER STRING	
CHR\$	CINT	CLEAR	CLOSE
CLS	COLUMN	COMMAND	COMMON
CONDITIONAL BRANCH		CONT	
CONTINUE PROGRAM		CONTROL KEYS	CONVERT DOUBLE
CONVERT INTEGER	CONVERT SINGLE	COS	COSINE
CREATE SUBSTRING		CSNG	CURRENT RECORD
CVD	CVI	CVS	DATA
DATA TO RANDOM BUFFER		DATE\$	
DECIMAL TO OCTAL		DEF FN	DEF USR
DEFDBL	DEFINT	DEFSNG	DEFSTR
DELETE	DELETE FILE	DIM	DROP FRACTION
EDIT	EFFICIENT MEMORY USAGE		ELIMINATE LINES
END	END FILE ACCESS	END OF FILE	EOF
ERASE	ERL	ERR	ERROR
ERROR BRANCH	ERROR CODE	ERROR LINE	ERRS\$
EXP	FIELD	FIX	FOR/NEXT
FORCE ERROR	FRE	FUNCTION	GET
GOSUB	GOTO	HEX\$	IF/THEN
INKEY\$	INP	INPUT	INPUT#
INPUT\$	INSTR	INT	KEYBOARD SCAN
KILL	LAST RECORD	LDOS IN BASIC	LEFT\$
LEN	LET	LINE INPUT	LINE INPUT#
LINE PRINT	LINK PROGRAMS	LIST	LLIST
LOAD	LOAD PROGRAM	LOC	LOF
LOG	LOG BASE E	LPOS	LPRINT
LSET	MATRIX VARIABLES		MEM
MERGE	MID\$	MKD\$	MKI\$
MKS\$	MULTIPLE BRANCH	MULTIPLE BRANCH	SUBS
NAME	NATURAL EXPONENT		
NATURAL LOGARITHM		NEW	OCT\$
ON ERROR GOTO	ON/GOSUB	ON/GOTO	OPEN
OPTION BASE	OUT	OVERLAYS	PACK DOUBLE
PACK INTEGER	PACK SINGLE	PASSING VALUES	PEEK
POKE	POS	POSITIONAL PRINT	
PRINT @	PRINT TAB	PRINT USING	PRINT#
PRINTER TAB	PUT	RANDOM	
RANDOM FILE BUFFER		READ	READ MEMORY
READ PORT	READ RANDOM FILE		READ TIME
READY FILE	REM	RENAME FILE	RENUM
RENUMBER LINES	REPLACE SUBSTRING		RESTORE
RESUME	RETURN	RIGHT\$	RND
ROW	RSET	RUN	SAVE
SGN	SIN	SINE	SPACES\$
SPC	SPECIAL CHARACTERS		SPECIFIED LOOPS
SQR	SQUARE ROOT	START PROGRAM	STATEMENT
STOP	STR\$	STRING LENGTH	STRING\$
SUBSTRINGS	SUSPEND EXECUTION		SWAP
SYMBOLS	SYNTAX	SYSTEM	
SYSTEM ERROR CODE		TAB	TAN

## Appendix C - - Keywords in the 6.1 Data Files

### Directory for HELP file : BASIC (cont)

TANGENT	TIME\$	TROFF	TRON
TRUNCATE	TRUNCATE STRING	UNCONDITIONAL	BRANCH
UNPACK DOUBLE	UNPACK INTEGER	UNPACK SINGLE	USER FUNCTIONS
USR	VAL	VARPTR	VERTICAL TAB
WAIT	WHILE/WEND	WRITE	WRITE MEMORY
WRITE PORT	WRITE PROGRAM TO	PRINTER	
WRITE PROGRAM TO SCREEN		WRITE RANDOM	
WRITE SEQUENTIAL		WRITE#	

### Directory for HELP file : LDOS

APPEND	ATTRIB	AUTO	BACKUP
BOOT	BUILD	CLICK/FLT	COM/DVR
COMM	CONV	COPY	CREATE
DATE	DEBUG	DEVICE	DIR
DO	DUMP	ERROR	FILTER
FLOPPY	FLOPPY/DCT	FORMAT	FORMS
FREE	HARD DRIVE	HARD FORMAT	JOBLOG
KSM/FLT	LIB	LINK	LIST
LOAD	MEMDISK	MEMORY	PASSWORDS
PATCH	PURGE	REMOVE	RENAME
REPAIR	RESET	ROUTE	RUN
SET	SETCOM	SETKI	SPOOL
SYN	SYNTAX	SYS ALIVE	SYS BLINK
SYS BREAK	SYS BSTEP	SYS DATE	SYS DRIVE
SYS FAST	SYS GRAPHIC	SYS RESTORE	SYS SLOW
SYS SYSRES	SYS SYSTEM	SYS TIME	SYS TYPE
SYSGEN	TAPE100	TIME	TRANSFORM4/CMD
TRSHD4/DCT	VERIFY		

### Directory for HELP file : TECH1

CMD FORMAT	DCB	DCB QR	DCB+00
DCB+01,02	DCB+03,05	DCB+06,07	DCT
DCT QR	DCT+00	DCT+01,02	DCT+03
DCT+04	DCT+05	DCT+06	DCT+07
DCT+08	DCT+09	DEVICE CONTROL	BLOCK
DIR QR	DIR+00	DIR+01	DIR+02
DIR+03	DIR+04	DIR+05,12	DIR+13,15
DIR+16,17	DIR+18,19	DIR+20,21	DIR+22,23
DIR+24,25	DIR+26,27	DIR+28,29	DIR+30
DIR+31	DIREC	DIRECTORY RECORDS	
DISK COMMAND FILE FORMAT		DISK I/O TABLE	
DRIVE CODE TABLE		EXTENDED DIRECTORY RECORDS	
FCB	FCB QR	FCB+00	FCB+01
FCB+02	FCB+03,04	FCB+05	FCB+06
FCB+07	FCB+08	FCB+09	FCB+10,11
FCB+12,13	FCB+14,15	FCB+16,19	FCB+20,23
FCB+24,27	FCB+28,31	FILE CONTROL	BLOCK
FILTERS & DRIVERS		FXDE	GAT
GAT QR	GAT+00,5F	GAT+60,BF	GAT+C0,CA
GAT+CB	GAT+CC	GAT+CD	GAT+CE,CF
GAT+D0,D7	GAT+D8,DF	GAT+E0,FF	
GRANULE ALLOCATION TABLE		HASH INDEX TABLE	
HIT	LOAD MODULE FORMAT		MEMORY HEADER
TAPE FILE OBJECT CODE FORMAT			



## Appendix C - - Keywords in the 6.1 Data Files

Help System Display Mode Version 6.1  
Copyright (c) 1983 by Logical Systems, Inc.

Directory for HELP file : TECH2

@ABORT	@ADTSK	@BANK	@BKSP
@BREAK	@CHNIO	@CKDRV	@CKEOF
@CLOSE	@CMNDI	@CMNDR	@CTL
@DATE	@DCINIT	@DCRES	@DCSTAT
@DEBUG	@DECHEX	@DIRRD	@DIRWR
@DIV16	@DIV8	@DODIR	@DSP
@DSPLY	@ERROR	@EXIT	@FEXT
@FLAGS	@FNAME	@FSPEC	@GET
@GTDCB	@GTDCCT	@GTMOD	@HDFMT
@HEX16	@HEX8	@HEXDEC	@HIGH\$
@INIT	@IPL	@KBD	@KEY
@KEYIN	@KLTSK	@LOAD	@LOC
@LOF	@LOGGER	@LOGOT	@MSG
@MUL16	@MUL8	@OPEN	@PARAM
@PAUSE	@PEOF	@POSN	@PRINT
@PRT	@PUT	@RAMDIR	@RDSEC
@RDSSC	@RDTRK	@READ	@REMOV
@RENAME	@REW	@RMTSK	@RPTSK
@RREAD	@RSLCT	@RSTOR	@RUN
@RWRT	@SEEK	@SEEKSC	@SKIP
@SLCT	@SOUND	@STEPI	@TIME
@VDCTL	@VER	@VRSEC	@WEOF
@WHERE	@WRITE	@WRSEC	@WRSSC
@WRTK	ERROR 00	ERROR 01	ERROR 02
ERROR 03	ERROR 04	ERROR 05	ERROR 06
ERROR 07	ERROR 08	ERROR 09	ERROR 10
ERROR 11	ERROR 12	ERROR 13	ERROR 14
ERROR 15	ERROR 16	ERROR 17	ERROR 18
ERROR 19	ERROR 20	ERROR 21	ERROR 22
ERROR 23	ERROR 24	ERROR 25	ERROR 26
ERROR 27	ERROR 28	ERROR 29	ERROR 30
ERROR 31	ERROR 32	ERROR 33	ERROR 34
ERROR 35	ERROR 36	ERROR 37	ERROR 38
ERROR 39	ERROR 40	ERROR 41	ERROR 42
ERROR 43	ERROR 43	ERROR 41	ERROR 42
ERROR 43	ERROR 63	ERROR DICTIONARY	
FLAGS USED	S0	S1	S10
S100	S101	S102	S103
S104	S11	S12	S13
S14	S15	S16	S17
S18	S19	S2	S20
S21	S22	S24	S25
S26	S27	S29	S3
S30	S31	S32	S33
S34	S35	S4	S40
S41	S42	S43	S44
S45	S46	S47	S49

## Appendix C - - Keywords in the 6.1 Data Files

### Directory for TECH2 (CONT)

S5	S50	S51	S52
S53	S54	S55	S56
S57	S58	S59	S6
S60	S61	S62	S63
S64	S65	S66	S67
S68	S69	S7	S70
S71	S72	S73	S74
S75	S76	S77	S78
S79	S8	S80	S81
S82	S83	S85	S87
S88	S9	S90	S91
S93	S94	S96	S97
S98	S99	SUPERVISORY CALLS	
SVC	UNKNOWN ERROR CODE		X0
X1	X10	X11	X12
X13	X14	X15	X16
X18	X19	X1A	X1B
X1D	X1E	X1F	X2
X20	X21	X22	X23
X28	X29	X2A	X2B
X2C	X2D	X2E	X2F
X3	X31	X32	X33
X34	X35	X36	X37
X38	X39	X3A	X3B
X3C	X3D	X3E	X3F
X4	X40	X41	X42
X43	X44	X45	X46
X47	X48	X49	X4A
X4B	X4C	X4D	X4E
X4F	X5	X50	X51
X52	X53	X55	X57
X58	X5A	X5B	X5C
X5E	X6	X60	X61
X62	X63	X64	X65
X66	X67	X68	X7
X8	X9	XA	XB
XC	XD	XE	XF

Z80A/HLP and Z80B/HLP have the same keywords as the 5.1 version.



# IFC - Interactive File Control



(PRO-)IFC/CMD: Copyright 1983/84 Karl A. Hessinger, All rights reserved.  
Published by MISOSYS, Inc., Alexandria, Virginia.

## Table of Contents

General Information . . . . .	1
Distribution Diskette . . . . .	1
Invoking IFC/CMD . . . . .	2
Invoking IFCLIST/CMD . . . . .	6

Note: LDOS is a trademark of Logical Systems Incorporated  
TRSDOS is a trademark of Tandy Corp.

## GENERAL INFORMATION

Anytime you have more than one disk drive on-line, you develop a collection of disks containing many files. File maintenance of moving files among disks while purging unneeded files can become a clumsy series of DIR commands followed by COPY and or KILL/REMOVE commands. Using PURGE with its file-by-file query can sometimes be useless when you have forgotten the contents of the files - you need to list them. IFC gives you the ability to perform these maintenance tasks interactively. A menu-controlled screen provides the tools to easily list, copy, delete, or rename a file or groups of files. You will find IFC essential to the task of file maintenance.

## DISTRIBUTION DISKETTE

This documentation covers the operation of both the LDOS 5.1 Model I/III version (IFC) and the LDOS or TRSDOS 6.x compatible version (PRO-IFC). The IFC package is provided on a 35-track single density data diskette for LDOS Version 5.1. The PRO-IFC package is provided on a 40-track double density data diskette for LDOS/TRSDOS Version 6. The diskette label identifies the DOS that the package is designed to function with.

## IFC - Interactive File Control

### INVOKING IFC

IFC is easily invoked via the command syntax:

```
=====
| IFC :d (X)
|
| d      - Specifies the logical drive number to work with.
|         The entry is optional.
|
| X      - Allow the use of IFC without a system disk in
|         drive 0.
|
=====
```

If no drive number was entered on the command line, IFC will prompt you for the working drive with:

**Select drive ( 0 - 7 ) ?**

Enter the number of the drive you wish to work with. After the disk directory has been scanned and its contents sorted, the following information will be displayed:

**FILENAME/EXT:D \*IP+ MM/DD/YY xxxxK :**

The first column is the file specification. The asterisk [\*] indicates that a file is a partitioned data set (PaDS). The uppercase "I" and "P" stand for invisible and protected file respectively. The plus sign [+] indicates that the file has been modified since the last time it was backed up. The third column contains the date on which the file was last modified. The last column is the file size rounded to the nearest K. This is not the amount of space that the file takes up on the disk but rather the size of the file. If the file had been tagged, then the colon would be followed by an asterisk to indicate its flagged state.

The first file on the screen will have the character string "==" pointing to it. We will call this string an "arrow". The file pointed to by the arrow will be referred to as the "current file". Any command which acts upon a single file will act on the file pointed to by this arrow. The copy, delete, tag and untag commands all affect the current file. To move this arrow, simply use the <DOWN ARROW> to advance to the next file in the list, or the <UP ARROW> to go to the previous file in the list. You may also depress the <SPACE BAR> to advance the pointer to the next file.

When the pointer advances to the start or the end of the list it will "wrap-around" to the end or the beginning. IFC will also display a blank line to indicate that wrap-around has occurred.

Depress the <H> key and IFC will display the help menu. The help menu looks like this:



## IFC - Interactive File Control

A - Again, retag files	C - Copy current file
O - <Un>Tag old/new files	D - Delete current file
T - Tag current file	L - List current file
U - Untag current file	R - Rename current file
W - Wildcard <un>tag	
+I*P - <Un>Tag by attributes	E - Exit to DOS
== <M>ass functions ==	F - Free space on drive
C - Copy files	H - Display help
D - Delete files	Q - Execute DOS command
R - Rename files	S - Select new drive

Press any key to continue

Any of IFC's commands may be aborted by depressing the <BREAK> key.

### CURRENT FILE COMMANDS

The following group of commands all deal with a single file. All of these commands will act on the current filespec.

#### <C>opy a file

This command will copy the current file to a specified drive. Depressing the <C> key will generate the prompt, **"Copy file(s) to drive ?"** Once you select the destination drive, the prompt, **"Reset MOD flags (Y/N) ?"** will be displayed. Depress <Y> and IFC will reset the modification flag in the directory of the source disk. Once the copy has been completed IFC will advance the arrow pointer to the next file.

#### <D>elete a file

Depress the <D> key and IFC will provide you with an opportunity to escape via the prompt, **"OK to delete file : filespec/ext:d ?"** Depress the <Y> key to delete the current file. Depress the <N> key or <BREAK> to abort. Once the file has been deleted, IFC will advance the arrow to the next file.

#### <L>ist a file

Depress the 'L' key and IFC will prompt, **"List file in ASCII (Y/N) ?"** Depress the <N> key to list the file in hex, or <Y> to list the file in ASCII. For the list function to work properly, you must have IFCLIST/CMD on a disk which is currently in one of your disk drives. If IFCLIST/CMD cannot be found, then the error message, **"Program not found"** will be displayed.

#### <R>ename a file

Press the <R> key and IFC will prompt, **"New name for FILESPEC/EXT:D ?"** Type in the new filename for that file and depress the <ENTER> key. The file will be renamed. The arrow pointer will be returned to the start of the list.

## IFC - Interactive File Control

### TAGGING COMMANDS

When an IFC command is given that will work on a group of files, the files must be tagged.

#### <T>ag files

Depress the <T> key and IFC will tag the current file. An asterisk [\*] will be displayed next to the file to indicate that the file has been tagged. IFC will also display a running total (in K) of all tagged files.

#### <U>ntag files

Depress the <U> key and IFC will untag the current file. The untag command works just like the tag file except in reverse.

#### <A>gain, retag files

Depress the <A> key and IFC will tag all of the files that were tagged and later mass copied. These files will be marked with a number sign [#] to indicate that the file was tagged and then copied.

#### <O>ld tag

This tagging command references the working disk against a target disk. Depress the <O> key and IFC will prompt you with, "<O>ld or <N>ew files ?" Depress the <O> key and all of the files which exist on both disks will be selected. If you depress the <N> key, all of the files which do not exist on the target disk but are in the currently selected disk will be selected. You will then be prompted with, "<T>ag or <U>ntag files ?" Depress the <T> key and all of the selected files will be set up for tagging, or depress the <U> key and the selected files will be set for untagging. Finally, you will then be prompted for the target drive with, "Drive to scan ( 0 - 7 ) ?" Depress the number of the drive to scan for the target drive. Your selection will then be invoked.

#### <W>ildcard tag

The wildcard tag command is used to tag or untag a group of files that match up with a wildcard file specification. Depress the <W> key and IFC will prompt with, "<T>ag or <U>ntag files ?" Depress <T> to have all matching files tagged, or <U> to have all matching files untagged. IFC will then prompt, "Filename wildcard ?" Enter a wildcard and IFC will either tag or untag all matching files. The wildcard file name and extension fields may consist of standard filespec characters [A-Z,0-9], question marks [?] which match all characters in their respective position, and an asterisk [\*] which will force a match of all characters to the end of the field.

#### <+IP\*> Tag by attribute

The modify tag will tag all files which have the modified flag set in the directory. These files will be marked with a '+' on the display.



## IFC - Interactive File Control

### MASS COMMANDS

The Mass commands are used to act on all of the tagged or all of the untagged files. Depressing the <M> key will begin the selection of a "mass" command. IFC will prompt, "**Mass <C>opy, <D>elete or <R>ename ?**" Make the selection by pressing the first letter of the function you wish to perform.

#### <M>ass copy

The mass copy command works just like a multiple copy command. IFC will move all tagged files or all untagged files automatically. The selection is made by responding to the prompt, "**Copy tagged or untagged (T/U) ?**" Depress the <T> key to select all of the tagged files, or the <U> key to select all of the untagged files. Don't forget that <BREAK> will abort the operation. IFC will then prompt you for the target drive with, "**Copy file(s) to drive?**" Enter the destination drive for the files you wish to copy or depress <BREAK> to abort. IFC then prompts, "**Reset MOD flags (Y/N) ?**" Answer <Y> and IFC will reset the modification flags in the source directory. IFC automatically resets the modification flags in the destination directory but you may not want them cleared from the source directory. Depressing <BREAK> will abort the mass copy function.

#### <M>ass delete

For mass delete, IFC prompts, "**Delete tagged or untagged files (T/U) ?**" Depress the <T> key and IFC will purge all tagged files from the disk. If you depress the <U> key, IFC will purge all of the untagged files. Depressing <BREAK> will abort the mass file delete operation.

#### <M>ass rename

The mass rename operation lets you change the file name and/or extension for all of the tagged files. The new filespec is derived by passing the current filespec through a template. The template is entered in response to the query, "**Rename template ?**" and is identical in syntax to the wildcard identified above. Alphanumeric template characters will be passed to the new filespec. The question mark [?] will cause the correspondingly positioned character of the current filespec to be part of the new filespec. If the template contains an asterisk [\*], the remaining part of the field of the current filespec will be transferred to the new filespec.

### MISCELLANEOUS COMMANDS

#### <E>xit to DOS

Depress the <E> key and IFC will return to DOS Ready.

#### <F>ree space

Depress <F> and IFC will prompt, "**Free space on drive ?**" Depress the number of the drive you wish IFC to scan. IFC will display the following information:

Drive : D      Disk name : NNNNNNNN      Free space : xxxxxK

IFC Utility - Invoking IFC

## IFC - Interactive File Control

where "D" is the drive number, "NNNNNNNN" is the disk's name, and "xxxxx" is the total amount of free space on that diskette. Depress <ENTER> to continue.

### <H>elp

Depress the <H> key and IFC will display the list of all IFC commands. Press any key to continue.

### <Q> Execute DOS command

Depress the <Q> key and IFC will prompt for the DOS command with the query, **"Enter DOS command:"** Enter any DOS command and IFC will execute the command. You should be careful not to perform any DOS command which will change the contents of HIGH\$ as IFC will protect itself above HIGH\$ when executing the DOS command. When the command has completed, IFC will resume after you respond to the prompt, **"Press any key to return to IFC..."**

### <S>elect new drive

This command is used to change the working drive. Depress the <S> key and IFC will prompt, **"Select drive ( 0 - 7 ) ?"** Depress the number of the drive you wish to log in. Depressing <BREAK> will return you to the command prompt. If, for some reason, IFC cannot log in the new drive, it will again display the Select drive prompt. Once IFC has returned to the Log drive prompt for the second time, a <BREAK> will return you to DOS Ready.



## IFC - Interactive File Control

### INVOKING IFCLIST

This program is used to perform the list function of IFC. It produces a screen-paged display. IFCLIST is invokable apart from IFC via the syntax:

```
=====
IFCLIST filespec (parm, parm...)
```

```
----- ASCII listing -----
```

```
NUM          Number lines. Defaults to OFF.
```

```
TAB          Expand tabs on output. Defaults to ON.
```

```
P           Send output to printer. Defaults to OFF.
```

```
PAUSE        Stop after displaying a screen of info.
              Defaults to ON.
```

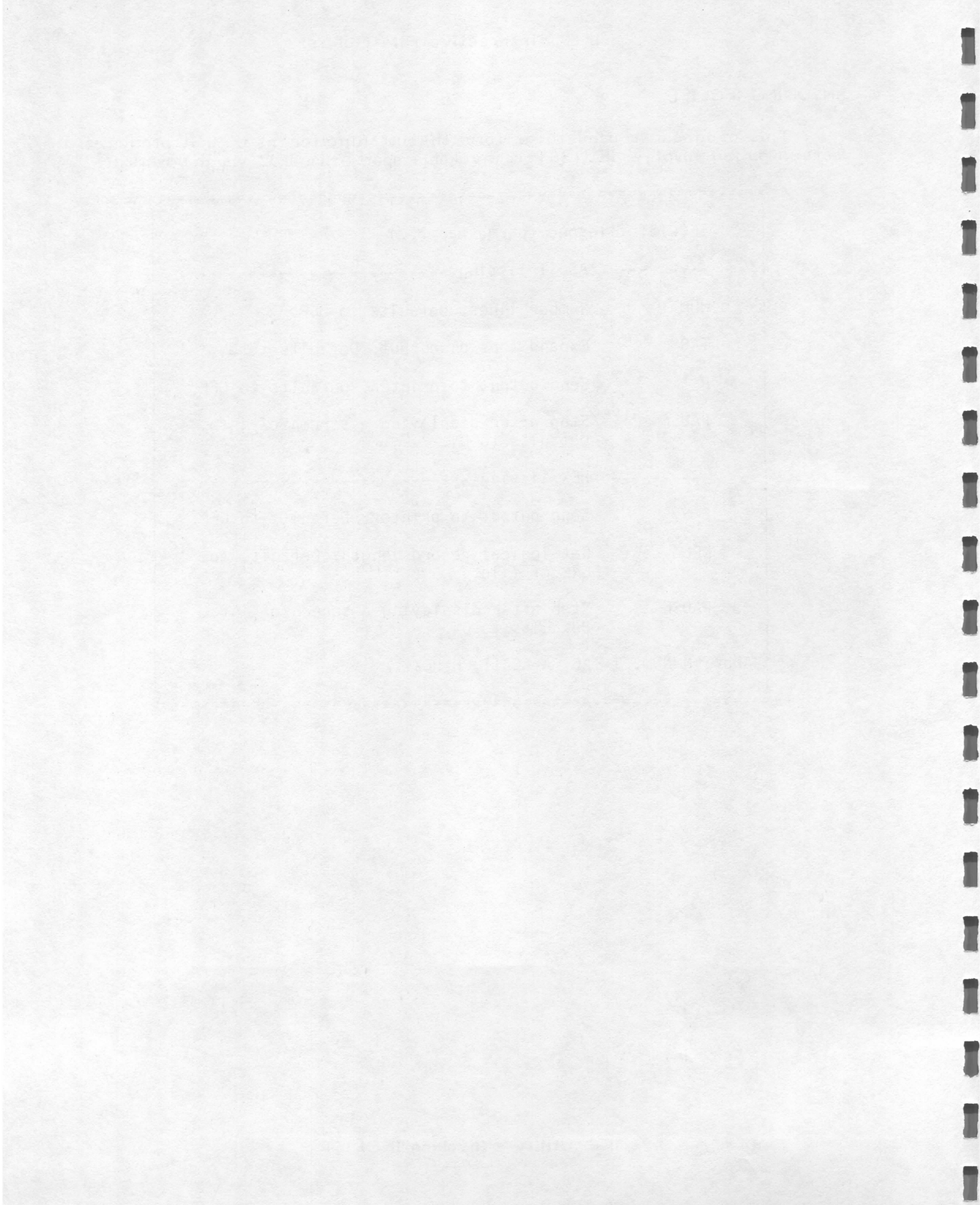
```
----- HEX listing -----
```

```
P           Send output to printer. Defaults to OFF.
```

```
LRL          Set logical record length. Defaults to
              LRL of file.
```

```
PAUSE        Stop after displaying a screen of info.
              Defaults to ON.
```

```
Abbr: N=NUM, T=TAB, A=ASCII, H=HEX
```





## OVERDRIVE

OVERDRIVE is a utility designed to provide "buffering" for a disk drive using part of an alternate memory bank. Furthermore, System Modules 1, 2, 3, 4, 5, 9, 10, 11 and 12 will be loaded into the alternate memory bank, and will produce results similar to those gained after executing a SYSTEM (SYSRES). It can ONLY be used on Model 4's and 4P's that have 128K of RAM. Up to two drives may be buffered. When a disk read is requested from a buffered drive, the entire track will be read into banked memory. Additional read requests for that track will transfer the data from memory, eliminating the physical disk access. When a system overlay is needed, it will be retrieved from alternate memory, again eliminating a physical disk access. In many cases this will greatly increase the speed of disk I/O. The syntax is:

```
=====
OD (parm,parm)

Optional Parameters:

B1=x   B2=y   Specify the drive(s) to be buffered,
               where x and y represent the drive number.
               Either or both may be specified. If
               neither is used, prompts will appear for
               the drives to be buffered.

REMOVE    When used with B1 and/or B2, will disable
           the buffering of the specified drive(s).
           The disk driver and system overlays will
           remain in memory.
           When used by itself (without B1 or B2)
           all drive buffering will be disabled, the
           back bank will be released, and the
           memory used by the driver will be freed
           (if possible).

abbr: REMOVE=R
=====
```

## I M P O R T A N T   N O T I C E

OVERDRIVE requires the TRSDOS 06.02.00 Operating System for proper operation.

### OVERDRIVE Installation

OVERDRIVE requires one free bank of alternate memory (16K for the drive "buffer" and 16K for the system overlays) and approximately 400 bytes of space in low memory for the disk driver. If there is not sufficient low memory or no back bank is available, OVERDRIVE will display an error message and the installation will be aborted. If OVERDRIVE is to be used with other "low" memory drivers/filters (e.g. COM/DVR or FORMS/FLT), it should be installed first to ensure that enough "low" memory is available.

When OVERDRIVE is installed, if System Modules have previously been resided in main memory (via SYSTEM (SYSRES)), an informative message will appear and no System Modules will be loaded into the back bank. This will not affect drive buffering, but will waste memory (the memory taken up by the SYSTEM (SYSRES) and the memory in the back bank reserved for the System Modules). When OVERDRIVE is active, you will not be allowed to SYSGEN (if OVERDRIVE is "trapped" in memory but not active, you will be allowed to SYSGEN - See Removing OVERDRIVE).

When OVERDRIVE is entered without specifying either B1 or B2, this prompt will appear:

Enter drive to buffer, <ENTER> to end

At this time, either 0, 1 or 2 drives may be specified for "buffering". After the first drive number is entered the prompt will appear again, and the second "buffered" drive may be specified (if desired). If <ENTER> is pressed and no drives are specified, the driver will be installed in low memory, the alternate memory bank will be allocated and the system overlay modules will be placed in the back bank, but no drive buffering will be done. This is useful for "reserving" low memory space for the driver.

During the initial installation of OVERDRIVE (i.e. if the driver is not already present), the <BREAK> key may be used to terminate the installation. The driver will NOT be installed in this case, and the back bank will not be allocated.

B1 and B2 may be used on the command line to enable disk buffering on one or two drives. Installation of OVERDRIVE when using B1 and B2 is identical to installing OVERDRIVE with no parameters, except there will be no drive prompts.

If OVERDRIVE has been installed and fewer than 2 drives are buffered, additional buffered drives may be specified at a later time. This is done in the same manner as initial installation.

Only drives that are enabled may be buffered. If you attempt to buffer a disabled drive, an appropriate error message will be displayed.

#### Removing OVERDRIVE

The buffering operation for a specific drive may be disabled by using a "drive" parameter (Either or Both B1 and B2) along with the REMOVE parameter. This will not affect the buffering of "non-removed" drives, and will not affect the use of the system overlays from the back bank.

Using the REMOVE parameter without specifying B1 or B2 will disable all drive buffering, release the alternate memory bank and attempt to remove the low memory driver. Informative messages will appear showing the operations which were performed.

If OVERDRIVE is "trapped" in memory, the alternate memory bank will be released but the "trapped" memory taken up by OVERDRIVE will not be "freed up". If this is the case, OVERDRIVE will occupy the same "trapped" memory allocation if later re-installed.

#### When to use OVERDRIVE

OVERDRIVE can greatly speed up processing of information that is read from a disk (especially a floppy disk). There will be a speed increase in all systems by having the system modules "resident" in the back bank. This will eliminate a disk access on any system overlay request. Please note that the "Library" modules (SYS6, 7 and 8) cannot be resided, and entering a library command (such as DIR) will cause a physical disk access.

The greatest increase in speed will result when a disk file is being accessed sequentially from a "buffered" drive (e.g. Record 1 is read first, followed by Record 2, Record 3, etc.). If a file's records are being read in a random order, drive buffering should not be enabled, as it may actually slow down processing (OVERDRIVE can still be used to reside the system overlays).

Library commands and utilities that do their own buffering (such as COPY and BACKUP) will not show a speed increase when used with OVERDRIVE (but in most cases will not show a decrease in speed either).



In certain cases, it may be beneficial to buffer drive 0 (e.g. when reading data sequentially from drive 0). However, for the more "routine" operations (such as executing TRSDOS Library and Utility commands), buffering drive 0 may slow down overall operation.

### SPECIAL CAUTIONS

If OVERDRIVE is to be used with "other" disk drivers (such as LS-DiskDisk, MemDisk or a hard disk driver), OVERDRIVE should be the LAST driver installed on the drive (do not confuse this with installation - OVERDRIVE may be installed in memory without having any drives buffered). For example, if you wish to use OVERDRIVE with LS-DiskDisk, install the DiskDisk driver first, followed by OVERDRIVE.

There are some things that should NEVER be done when using OVERDRIVE. These are:

- 1) Performing the TRSDOS Library command SYSTEM (SYSTEM=n) if either drive 0 or drive n is buffered. If you need to change your "System" drive, you MUST disable the buffering for BOTH drives before doing so.
- 2) Buffering both the "outer" and "inner" drive when LS-DiskDisk is used (Note: The "outer" drive is the actual physical drive. The "inner" drive is the DiskDisk file which is acting as a drive). Either the outer drive or the inner drive may be buffered. Buffering BOTH at the same time will cause unpredictable results (more than likely a system "hang up" when the inner drive is accessed).
- 3) Performing a DEBUG Read or Write operation on a buffered drive. OVERDRIVE uses the same area of memory that DEBUG uses when a read/write operation is done. Using DEBUG to read/write a buffered drive may cause a system "crash".

### Technical Notes

One final word about OVERDRIVE concerns the "logging" in of a new diskette in a buffered drive. When accessing the directory (or track 0), no buffering will be done, and the "current" buffered information will be marked as "old". This will force the next "read" to perform physical I/O, and will prevent any "old" buffered information from being used. In most cases, diskette swapping will present no problems.

### Examples

The following command could be used to install OVERDRIVE and enable buffering on drives 2 and 4.

OD (B1=2,B2=4)

If drives 2 and 4 are currently buffered and you wish to disable the buffering for drive 4, this command could be used:

OD (B1=4,REMOVE)

To disable all drive buffering, release the back bank and attempt to de-allocate the memory used by OVERDRIVE, use this command:

OD (R)

### I M P O R T A N T   N O T I C E

The OVERDRIVE package is a product of Logical Systems, Inc. It has been designed and tested to work with the LS-DOS/TRSDOS 6.2 operating system. The LS-DOS/TRSDOS 6.2 operating system is a product of Logical Systems, Inc., and is licensed to Tandy Corporation. TRSDOS is a trademark of Tandy Corporation.

This package is sold on an "as-is" basis. Logical Systems, Inc. makes no expressed or implied warranty of any kind with regard to the software or documentation. Under no circumstances will Logical Systems, Inc. assume any liability for actual, incidental or consequential damages resulting from the use of this package. Furthermore, under no circumstances will Logical Systems, Inc. assume any liability for actual, incidental or consequential damages resulting from the use of the LS-DOS/TRSDOS 6.2 operating system.

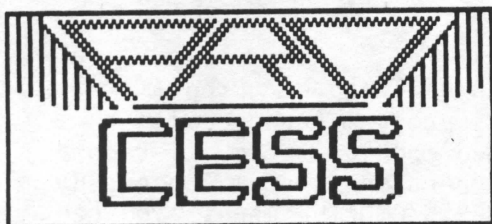
From time to time, updates to this product may become available for a nominal charge. Customer Service information on this product and any available updates may be acquired by contacting Logical Systems, Inc. at the following address:

Logical Systems, Inc.  
8970 N. 55th Street  
P.O. Box 23956  
Milwaukee, Wisconsin 53223

(414) 355-5454

The entire OVERDRIVE package and manual is Copyrighted 1984 by Logical Systems, Inc.





## TABLE OF CONTENTS

GENERAL . . . . .	1
WHAT IS PRO-CESS/CMDFILE? . . . . .	2
THE PRO-CESS/CMDFILE MENU . . . . .	3
RE-ENTRY AFTER INADVERTENT EXIT . . . . .	4
COMMAND DETAILS . . . . .	5
PRO-CESS/CMDFILE PUT TO USE . . . . .	12
DISK LOAD MODULE FORMATS . . . . .	13

Authored and copyrighted (C) 1979/1983 by Roy Soltoff.  
PRO-CESS/CMDFILE is published by MISOSYS, Alexandria, VA.

LDOS is a trademark of Logical Systems, Inc.  
TRS-80 and TRSDOS are trademarks of Tandy Corp.

### GENERAL

=====

PRO-CESS/CMDFILE is a powerful maintenance tool for "CMD" or "CIM" type load module files. It provides for file appending, mapping, sorting, packing, offsetting, library member and partitioned data set member extraction, as well as the specified deletion of any load module record. PRO-CESS/CMDFILE can convert "CMD" files which contain various types of records to "CIM" files which are pure binary core-image constructs. It also provides the capability of converting "CIM" files to "CMD" files. PRO-CESS/CMDFILE gives capabilities to load module maintenance never before possible. PRO-CESS/CMDFILE does it all: rapidly, totally, and economically!

The CMDFILE utility is provided on a 35-track single density data diskette and is operational on TRS-80 Models I and III. The PRO-CESS utility is provided on a 40-track single density data diskette and is operational under LDOS/TRSDOS Version 6.

## WHAT IS PRO-CESS/CMDFILE?

=====

The PRO-CESS/CMDFILE tool is a powerful machine language program that has been designed to provide total maintenance of program load modules on a record basis. This means that it references the load module as a multi-record type, variable length record file - just like the operating system loader. By using the various commands identified in the PRO-CESS/CMDFILE menu, you can completely reorganize the load structure of a given module or modules in order to make them more efficient in terms of loading speed and occupied disk space.

PRO-CESS/CMDFILE also provides the capability of converting "CMD" type load modules to/from "CIM" core-image load modules. This is especially useful to generate program files in "binary" form for PROM burners. The "CIM" structure is identical to the "COM" structure used in other types of operating systems.

You get the capability of appending two or more "CMD" machine language load module files into one file. This is useful to concatenate two or more separately assembled OBJECT code files, concatenate two or more non-contiguous blocks of code, or also couple two or more programs together so they load together. You get control of the program's transfer address or ENTRY point.

"CMD" files can be copied from one SYSTEM diskette to another SYSTEM diskette on a single drive system provided both diskettes use the same operating system.

You get the capability of totally mapping every record in a load module. Determine the TYPE as well as the load address range of each load record. This load map is displayed in the MENU status. You can optionally request that a map listing be sent to a line printer.

You can selectively remove any record in the load module. Get rid of space-wasting headers that are not necessary. Remove dead space. In case you make a mistake, you can even un-remove a removed record before clearing the load-module's memory buffer.

By far, the most powerful function included in this tool is the reorganization capability of the PACK command. This powerful function converts any X-type patches to D-type patches [X-type patches are generated by the LDOS Version 5 or LDOS/TRSDOS Version 6 PATCH utility]. It then sorts the buffer by load address to construct sequential load records and generates a load module file that uses maximum sized (256-byte) load records. This feature is quite useful for reorganizing large inefficiently generated load modules such as Tandy's COBOL package. The PACK function is also useful for reorganizing the out-of-sequence load modules generated by the LC compiler [the records are out-of-sequence due to the separation of program and data regions during the compilation process].



## PRO-CESS / CMDFILE Load Module Maintenance Tool

### THE PRO-CESS/CMDFILE MENU

When you execute the PRO-CESS maintenance tool, all of its functions are immediately available through single letter commands. These commands are displayed in the PRO-CESS MENU. A reasonable facsimile of the MENU follows.

```
PRO-CESS 2.0 [Copyright (C) 1983 Roy Soltoff]

<C>lear the buffer region
<D>OS Command request
<E>xit to DOS
<I>mage file load/write
X<L>oad a file into the buffer
<M>ap the buffer records
<O>ffset address from current load origin
<P>ack the buffer records
<R>emove a record from the buffer
<S>ort the load records by address
<U>n-remove a "removed" record
<W>rite the buffer to a disk file

Buffer: Size 46802 Used 00002 Free 46800 Records 00000
Module: Origin FFFF End 0000 Entry 0000 Offset 0000

.....
```

This menu is designed for display on a 80x24 video display. The CMDFILE MENU contains exactly the same information but is compressed to fit on a 64x16 video display.

The menu of commands contains each command letter within angle brackets, "<>". For instance, the command to "<L>oad a file into the buffer" can be selected by depressing the "<L>" key on the keyboard. Each of the commands displayed in the MENU may be selected by depressing whatever key is contained in the angle brackets. Notice that the "<L>" command has a large blinking graphics block preceding it. This "cursor" is used for an alternate means of command selection. If you depress the <UP-ARROW> key, the block will move up to precede another command. The <DOWN-ARROW> key will move the block down to the next lower command. The block will wrap around in either direction. When PRO-CESS/CMDFILE is waiting for you to enter a command request, this cursor block will blink. When a command is being processed, the cursor block will not be blinking. Any menu command that is preceded by the graphics block can be selected just by depressing the <ENTER> key in lieu of the command letter. This feature is provided for your convenience. When you use the command letter to select a command, the block will be automatically moved to that menu command so you will have a visual indicator as to the command currently in progress.

In order to provide maintenance on a load module file, the file must be loaded into the PRO-CESS/CMDFILE memory buffer. The MENU will constantly display the status of this buffer via the status line titled, "Buffer:". The status line contains four fields: Size, Used, Free, and Records. The "Size"

## PRO-CESS / CMDFILE Load Module Maintenance Tool

field will display the total number of bytes provided for the buffer. This value is determined from the memory available between the end of the PRO-CESS/CMDFILE program and the highest memory address available based on the "HIGH\$" value. The "Used" field contains the number of buffer bytes that are currently in use. Each record that is loaded into the buffer requires a two-byte linkage pointer in addition to the memory taken up to store the record. Two bytes are initially used to store the "head" linkage pointer. The "Free" field shows you how many bytes are available for use. This field is the difference between "Size" and "Used". The "Records" field maintains a count of the total number of records stored in the buffer.

The MENU also displays the status of the program load records currently stored in the buffer. This status covers the program's ORIGIN or lowest address, its END or highest address, its ENTRY or transfer execution address, and any OFFSET specified. The OFFSET is a maintenance function that can be used to construct a file which loads into an address space different from where it executes.

The line of dots represents a third status/prompt line which will display informational messages pertinent to PRO-CESS/CMDFILE commands and prompting messages where required. It is also used to display any error message returned by the operating system during service requests.

### RE-ENTERING PRO-CESS/CMDFILE AFTER INADVERTENT EXIT

=====

It sometimes happens that you <E>xit the program without <W>riting the buffer to a file. Rest assured that you can recover from this mishap. If you immediately execute:

PROCESS \* or CMDFILE \*

the program will not automatically <C>lear the buffer region. The asterisk, "\*", provides the needed error recovery. Use the <M>ap command to scan through the buffer to ascertain its validity before attempting to generate a file.



## COMMAND DETAILS

=====

The following sections will describe the function and operation of all PRO-CESS/CMDFILE commands identified in the MENU.

## &lt;C&gt;lear the buffer region

-----

The <C>lear command is simple enough - it restores the buffer as if you just executed the program. Since this action will automatically clear any load module contained in the buffer, PRO-CESS/CMDFILE gives you a second chance to acknowledge your selection. The MENU status will display the prompt:

Are you sure you want to clear the buffer <Y,N> ? >

By entering a response of <N>, you will abort the <C>lear selection. A <BREAK> will also abort the selection. Only by entering a <Y>, will the <C>lear function activate. When the buffer is cleared, you will observe a "flash" of the video display as the MENU is re-generated.

## &lt;D&gt;OS Command request

-----

The <D>OS Command function provides access to operating system commands from the MENU level. Your DOS requests should be limited to library commands [a summary of library commands is normally obtainable via the "LIB" DOS command]. You enter your command request in response to the prompt:

Command? >

After your command is entered, the MENU will be erased while your command line will be displayed at the top of the video display screen. When the command that you requested is completed, you must depress the <ENTER> key to refresh the MENU display. This provides the opportunity of analyzing any screen image displayed by the DOS command before the MENU display erases the image.

## &lt;E&gt;xit to DOS

-----

This command provides the means to terminate the maintenance session and return to DOS.

## &lt;I&gt;mage file load/write

-----

This command provides two functions - both relating to core-image files. Use the <I>mage command to load a core-image file from disk into the buffer. You also will use the <I>mage command to write the buffer out as a core-image file. To help with your selection, the MENU will display the prompt:

## PRO-CESS / CMDFILE Load Module Maintenance Tool

### Image file LOAD or WRITE <L,W> ? >

A response of <L> invokes the image loading function which subsequently requests you to enter the image file specification via the prompt:

#### Input file specification >

If you omit the file extension, PRO-CESS/CMDFILE will use "/CIM" as the default extension.

Since core-image files have no loading information contained in the file, it is necessary to specify the origin address of the module. You do this in response to the prompt:

#### Enter the module origin or <ENTER> to use [0000] >

Your selection must be entered in hexadecimal. Also, as can be noted by the prompt, if you depress just the <ENTER> key without a value, a default origin of 'X'0000' will be used. This value will also be used for the ENTRY or transfer address. The entire file, including the full last sector, will be considered as part of the program.

If you respond to the initial prompt with a <W>, you will invoke the image writing function. It is essential that the load records stored in the buffer be in sequential load order. The <IW> function will first scan the load records to ensure that they are, for a core-image file cannot be constructed if the records are out of order. If a problem is detected, the MENU will display the error message:

**Buffer is not in sequential load order!**

It is not necessary for the load records to be contiguous. The <IW> function will generate null bytes, 'X'00', for all addresses interstitial to two adjacent non-contiguous load records. You identify the file specification of the file to be written by responding to the MENU prompt:

#### Output file specification >

If you omit the file extension, the default value of "/CIM" will be used. Upon successful completion of the file generation, the message:

**Requested file now written**

will be displayed and PRO-CESS/CMDFILE will await your next MENU selection.

#### <L>oad a file into the buffer

-----  
The <L>oad function is used to read a load module file into the memory buffer. The file will be appended to any already contained in the buffer. You identify the specification of the file in response to the prompt:

#### Input file specification >



## PRO-CESS / CMDFILE Load Module Maintenance Tool

If you omit the file extension, the default value of "/CMD" will be used. While the file is being read into memory, PRO-CESS/CMDFILE analyzes it to determine the specific type of load module: OBJECT file, ISAM overlay file, or a partitioned data set (PDS) file [the PDS file structure has been defined by MISOSYS]. An object file will be loaded directly into the buffer.

If the file is recognized as an LDOS structured ISAM overlay file, you will need to identify the overlay number of the desired member. In general, system files SYS6/SYS and SYS7/SYS are both ISAM overlay files. Under LDOS Version 6, SYS8/SYS is also an ISAM overlay file. These numbers are listed later. You will be requested to enter this ISAM number entry with the prompt:

**File has ISAM overlays - enter # >**

This number is entered in hexadecimal. If you respond to the prompt by depressing the <ENTER> key without a number, the entire file will be loaded into the memory buffer. The only purpose for doing this would be to map the file as no reorganization is possible with this maintenance tool.

If the file is recognized as a PDS file, you need to specify a MEMBER specification. This is done in response to the prompt:

**File is a partitioned data set, enter MEMBER >**

The MEMBER is the eight-character member specification as observed from a directory display of the PDS members. If you respond to the prompt by depressing the <ENTER> key without a MEMBER, the PDS Front End Loader program will be loaded into the memory buffer. Since it is likely that any given PDS contains non-CMD members, the PRO-CESS/CMDFILE maintenance tool does not attempt to read the entire PDS file into memory.

If you specified an ISAM number or MEMBER that can not be located in the file, the error message:

**Requested ISAM member is not in the file!**

will be displayed. It is highly improbable to receive the error message:

**Overlay beyond end of file!**

however, if you do, it means that the ISAM directory contains a location for the member that is not within the scope of the file. You probably have an error in the file.

If any disk I/O error results while the file is being read, or any problem occurs that results in the file not being read to completion, PRO-CESS/CMDFILE will return to the MENU command request after displaying an appropriate error message. No fragment of the file will be added to the memory buffer.

If there is no more available space in the memory buffer during the loading of a file, the error message:

**Insufficient buffer space to load file!**

will be displayed.

If you attempt to load in a file that is not a load-module structure file, PRO-CESS/CMDFILE will display the error message:

**File is not a load module file structure!**

and the load operation will cease.

Upon successful completion of the load operation, the message:

**File is now loaded into the buffer**

will be displayed. The buffer and module status lines will be updated to reflect the revisions made to the buffer with the load of the file.

A <BREAK> detected during the loading of a file will immediately abort the loading operation. No fragment of the file will be added to the memory buffer.

#### <M>ap the buffer records

-----

This MENU command provides the function of mapping the buffer contents. Mapping is useful for obtaining the record number of records you wish to remove. It is also helpful to understand how unorganized your load module file is. The prompt message:

**MAP output to printer <Y,N> ? >**

gives you the option of directing a load-module map to a printer by responding with <Y>. If you respond with <N>, only the status line will display the mapping, one record at a time. If you do not select a printer map, it is necessary to depress the <ENTER> key to obtain the mapping information for each record.

Each record will be given a sequential logical record number. This number is used as a record reference in the <R>emove and <U>n-remove MENU commands. Records will be identified as to type: Module header, Yanked load block, Load, Transfer Address, and so forth. The address range of load records will also be displayed.

If you specified the printer option, the printer will first be checked for availability. If it is not ready for use, the message:

**Printer is not available!**

will be displayed until it is made ready. The <BREAK> key can be used to escape from this condition.

#### <O>ffset address from current load origin

-----

Offsetting a load module means changing its loading address so that it loads into memory at a location different from where it was assembled to



execute. There are a few reasons for wanting to offset a file. One, of course, is to offset a file assembled to run from a PROM so that it loads into a RAM region usable by a PROM burner. The <O>ffset MENU command requests the revised load origin via the prompt:

Enter the offset origin address >

This entry is to be made in hexadecimal. For example, if the existing load module origin is X'3000' and you want it to load starting at X'5300', enter the four-character value, <5300>.

<P>ack the bufferrecords  
-----

The <P>ack operation is the most powerful feature of PRO-CESS/CMDFILE. It is used to reorganize an object load module file so that it is most efficient in disk storage space and optimum for rapid loading by the operating system. Packing is a three-phase operation. The first phase identifies any LDOS X-type patch records and packs the object code revisions into the preceding load records wherever possible. Any patch address that is outside the range of the existing load records, is used to generate new load records. The second phase then uses the <S>ort facility to sequence the load records by sequential load address. The third and final phase generates a new object load module file with maximum-sized load records. This is achieved by combining short contiguous load records wherever possible.

The first two phases require no action from you. Appropriate status messages are displayed to apprise you of the phase. The first phase is identified by the message:

Packing any "X" patches...

The second phase is noted by the sorting message as noted in the <S>ort command discussion. The third phase will generate the dialogue as noted in the <W>rite command discussion.

<R>emove a record from the buffer  
-----

This MENU command can be used to delete an entire record from the load module. You must identify the record by number. The record's number can be identified with the <M>ap command. The record is deleted by setting a "removed" flag for the record which is bit-7 of the record's TYPE byte. For instance, a load record will be changed from TYPE=01 to TYPE=81. If you map the buffer after removing a record, you will observe the change. Any record that is "removed" will not be written to a file during the <W>rite or <I>mage <W>rite commands.

<S>ort the load records by address  
-----

This command reorganizes the buffer's load records [record type 01] so that they are in sequential load order. If non-load records are intermixed between the load records, they may not maintain their position after the

buffer is sorted.

If the buffer contains any X-type patch records that were generated by the LDOS utility, PATCH, the program may be adversely affected by sorting. To apprise you of this situation, the <S>ort command will display the message:

X-patches present. Do you still want to sort <Y,N> ? >

If the patch type-01 records are totally outside of the load range of all non-patch type-01 load records, you may procede to sort the buffer. If no patch type-01 record extends into the range of a non-patch type-01 record, you also may procede with the sort [this implies that a type-01 patch record is wholly contained within the range of a previous type-01 record]. If you are unsure of the consequences, do not sort. An alternative is to use the <P>ack command which packs any X-type patches into the non-patch area of the buffer space.

The sort operation will commence with the display of the message:

Buffer sort commencing...

If the buffer contains a very large file, the sorting may take a half-minute or more. A blinking asterisk will amuse you while you await the completion of the sort. Upon completion, the status message:

Buffer is now sorted

will be displayed. Note that the record numbers are changed if the sorting process detects any out-of-sequence load record.

<U>n-remove a "removed" record

-----

If you inadvertantly remove the wrong record with the <R>emove command, you can recover from your error with this <U>n-remove function. As previously stated, the record number is obtained from a buffer mapping operation.

<W>rite the buffer to a disk file

-----

This command is used to write the buffer contents to a disk file. Since you may have appended two or more modules into the buffer, you now have the opportunity of changing the module's ENTRY address as noted in the MODULE's status display. This capability is useful when appending two or more files since the transfer address used would default to the transfer address of the last file loaded. Respond to the prompt:

Enter new ENTRY address or <ENTER> to use [xxxx] >

If you want to change the transfer address (entry point), you can enter the new address in hexadecimal. If you want to maintain the ENTRY as specified in the MODULE's status line [and also repeated in the prompt], just depress the <ENTER> key.



## PRO-CESS / CMDFILE Load Module Maintenance Tool

You identify the file specification of the file to be written by responding to the MENU prompt:

### Output file specification >

If you omit the file extension, the default value of "/CMD" will be used. Upon successful completion of the file generation, the message:

### Requested file now written

will be displayed and PRO-CESS/CMDFILE will await your next MENU selection.

## PRO-CESS/CMDFILE PUT TO USE

=====

### Appending two or more files

-----

You may have the occasion to assemble two separate object files that you want to combine into one. In order to append or concatenate two or more files into one contiguous file, use the <L>oad command to combine the files into the memory buffer. When the last file has been loaded, use the <W>rite command to generate the combined object load module file. Note that the transfer address of the concatenated file would be the transfer address detected from the last file input. The <W>rite command provides the opportunity of modifying the transfer address to one of your choosing.

### Customizing an LDOS library with PaDS/PRO-PaDS

-----

Since PRO-CESS/CMDFILE provides the capability of extracting ISAM members from LDOS libraries, and PaDS/PRO-PaDS provides the capability of building USER libraries, we can combine the power of both utilities to customize a USER library with DOS library commands.

If you execute an LDOS LIB command, you will see LIB <A> and LIB <B> commands displayed. LDOS/TRSDOS 6.x users will also see LIB <C>. The names of each command represent the entries to members in SYS6 and SYS7 (also SYS8/SYS for LDOS 6.x LIB <C>) respectively. The command interpreter which resides in SYS1 compares your command entry to a table which contains ISAM numbers for each LDOS LIBRARY command. It is these numbers that are needed to extract one of the LIBRARY members. Here is a list of the numbers for each command:

SYS6-LIBA	SYS6-LIBA	SYS7-LIBB	SYS7-LIBB	SYS8-LIBC
-----	-----	-----	-----	-----
31-APPEND	41-LIST	51-ATTRIB	72-PURGE	B1-FORMS
32-COPY	81-LOAD	11-AUTO	A1-SYSTEM	B2-SETCOM
61-DEVICE	1E-MEMORY	33-BUILD	16-TIME	B3-SETKI
21-DIR	53-RENAME	17-CLOCK	1A-TRACE	A2-SPOOL
91-DO	63-RESET	13-CREATE	1B-VERIFY	
66-FILTER	64-ROUTE	15-DATE		
18-KILL*	82-RUN	14-DEBUG		
19-LIB	65-SET	71-DUMP		
62-LINK	A2-SPOOL	22-FREE		

\* LDOS 6.x command name is "REMOVE"

### Optimizing an inefficient load module

-----

You just purchased that new COBOL compiler and are perturbed at how long it takes to load - not to mention the disk space it takes up. When you load it into PRO-CESS/CMDFILE and perform a <M>apping, you are amazed to find that "RUNCOBOL" and "RSCOBOL" are generated with 16-byte load records. Use the <P>ack command to reorganize the inefficient RUNCOBOL file and shrink it from 1537 records to 97 records while at the same time you reduce the amount of disk space taken up by the file from 121 sectors (31.5K) to 98 sectors (25.5K) - a savings of 6K of disk space.



## DISK LOAD MODULE FORMATS

=====

A load module is simply a disk file that can be loaded into memory by the system loader. The file is made up of variable length records and is usually a program. Many different types of records are included in a load module - the DOS makes extensive use of distinct record types in load modules. One record type is a load record which contains information on where it is to load into memory. If the file can be directly executed as a program, it then becomes known as an executable load module (ELM). The usual term that has been applied to such a file is "CMD". That's because a directly executable load module can be invoked as if it were a system CoMmanD. We commonly use the file extension of "/CMD" for these command files.

A load module can be conceptualized as a sequence of RECORDS. Note that we did not say an ordered sequence. Thus, the implication is that the records do not have to be in an ascending order (contiguous load addresses). Each record contains three fields: a TYPE field, a LENGTH field, and a DATA field. It has a one-byte indicator as to what TYPE of record it is. This TYPE code is used to denote a record as a HEADER record, a TRANSFER record, an ISAM directory entry record, a LOAD record, or other meaningful structure. Each record also has a one-byte LENGTH field which is the length of the data area field. The data field length thus ranges from <1-256> in value [a 0 implies 256]. The remaining part of the record is its DATA AREA and is used to store program code, directory information, messages, or other pertinent information. If you are familiar with BASIC random access files, you will see the similarity in the fielding of records - except in this case, we have variable length sequentially accessed records [with partitioned data sets provided in the PaDS/PRO-PaDS utility, you also have variable length indexed sequential accessed records]. Figure 1 lists the various TYPE codes currently used in operating systems.

TYPE	DATA AREA
01	Object code load block
02	Transfer address
03	End of load-only program
04	End of partitioned data set member
05	Load module header
06	Partitioned data set header
07	Patch name header
08	ISAM directory entry
0A	End of ISAM directory
0C	PDS directory entry
0E	End of PDS directory
10	Yanked load block
1F	Copyright block

Figure 1: Load Module TYPE Codes

Any code above X'1F' is invalid as a record type. In addition, any code not listed in figure 1 is reserved for future use.

If you could look at a sample object program file, you would notice that it starts out with something like:

```
05 06 50 52 4F 43 45 53 1F 1E 43 6F ...
. . P R O C E S . . C o ...
```

stretched across the screen. What you have here is a load module header (TYPE=05). The length byte (LENGTH=06) follows the TYPE code. The 6-byte DATA AREA field is the header name. All records follow this "fielding" order. A record is organized with a TYPE, LENGTH, DATA sequence. The X'1F' begins the second record. It happens to be a copyright record with a LENGTH of X'1E' or 30 decimal bytes. Incidentally, the TYPE=1F record is generated automatically by the "COM" pseudo-op in EDAS/PRO-CREATE, the macro-assembler used to develop and maintain the LDOS operating system.

Note that each record begins with the TYPE code and the first byte following the end of a record is always the TYPE code of the next record. The only exception is when a TYPE code indicates the end of a file. If you look further in the record displayed at relative position X'28', or if you count 30 bytes down from the "C" of "Copyright", you will see:

```
01 02 00 30 ...
```

The record TYPE is a load block (TYPE=01), and the length of the data area is X'02', or 258 data bytes. Yes, we previously stated that the length ranged up to 256 and here we have 258! This TYPE=01 record is a special case. The two-byte field following the LENGTH is the starting load address for the rest of the field. Since the LENGTH value includes the 2-byte load address, a length of X'03' would indicate only one load byte. A length of X'04' would indicate two load bytes. A length of X'FF' would indicate 253 load bytes. A length of X'00' would indicate 254 load bytes. To be able to have a data area with up to 256 bytes of loadable data, the LENGTH values of X'01' and X'02' are indicative of 255 and 256 load bytes respectively. This is accomplished by having the system loader decrement the length value by two when reading a load address. The resultant value becomes the true length of the loadable data.

If you could look at the last four bytes of the file, they appear as:

```
02 02 00 30
```

This will represent the TRANSFER record (TYPE=02). Again, we have a LENGTH byte which shows a 2-byte data field. The data field contains the transfer address or entry point to the program in standard low-order, high-order sequence. The system uses this address as an entry to the program after successfully loading it into memory. This address is also what is returned in register pair HL by the @LOAD SuperVisor Call.

So far we have discussed the HEADER, the COPYRIGHT, the LOAD, and the TRANSFER records. These are the four common record types you will find in most load module files. We also observe that our discussion of program load modules was limited to a single program per file. Another kind of file is one that contains many program modules (or data modules) as sub-files. Since the file is divided into sub-files, it is considered a "partitioned data set" abbreviated as "PDS". The PDS contains a directory of its sub-files with each



sub-file being termed a MEMBER of the PDS and having an entry in the directory. The system loader supports a particular kind of PDS used to contain the library overlays used in LDOS.

If you could look at a library file, you would see something like:

08 06 21 00 24 00 00 CB 08 06 61 ...

The TYPE code of X'08' indicates an ISAM DIRECTORY ENTRY record. The LENGTH byte denotes a DATA area of six bytes. After the sixth byte, you will see another TYPE=08 starting another ISAM directory entry record. The file is a partitioned data set. The TYPE=08 records are the directory entries for its members.

The ISAM directory data area is used by the SYSTEM loader to locate where a particular member can be found in the file. The data area includes positioning information indicating the exact byte position in the PDS which is the first record of the member. The six-byte data field is further divided into sub fields. The first byte (in this case, X'21') is the ISAM entry number. This entry number is provided to the system loader when a library command is parsed by the command interpreter. The entry number is the PDS member that will execute your request. The system loader searches the PDS directory for a matching directory record. The next two-byte sub-field is the transfer address of the member. The transfer address is contained in the directory so that more than one transfer address can be applied to a member. Therefore, a member can have multiple entry points. The last three-byte field is the triad pointer which points to the first byte of the member. The triad pointer is composed of the Next Record Number (NRN) and Relative Byte Offset for the member's first record byte. The system then positions to the pointer and loads the member. Thus you have six bytes of data as specified by the LENGTH byte. Since the process uses an index (the directory) to locate the member's starting byte then proceeds to sequentially read the member, the access method is termed "Indexed Sequential Access Method" (ISAM).

A TYPE-08 record can also have a 9-byte data area. In the PaDS/PRO-PaDS utility available from MISOSYS, the ISAM directory entry record includes a three-byte subfield which contains the TRUE length of the member. The position of a member's logical end-of-file (EOF) can thus be calculated by adding its length to its position and adjusting for sector boundary alignment.

If you could look at the first byte following the last TYPE-08 record, you would observe the sequence:

0A 01 00 04 01 00 01 02 00 26 ...

The TYPE=0A indicates that it is the end of a PDS directory. The SYSTEM loader will return a "file not found" error if it reaches this record without finding a match of the ISAM number. The LENGTH=01 is needed because ALL load module records MUST have a length byte. The DATA area contains only a single arbitrary byte, X'00'. We cannot indicate a null record because a length byte of X'00' indicates 256 data area bytes. Thus, the X'0A' record type must have a minimum of one byte in its data area.

The following record is a TYPE=04 to indicate the end of a PDS member. This record serves but one purpose when used immediately following the

directory - it will result in the return of a "Load file format error" if a library file is executed as if was a CMD file. When not expecting a partitioned data set file, the SYSTEM loader will ignore record types other than X'01' and X'02' except for the X'04'. The file reading will terminate at the X'04' with the above-mentioned error message.

The record type X'04' is usually used at the end of each partitioned data set member. Each member will usually end with "04 01 00" rather than a TYPE=02 record. The system loader uses the X'04' type code in lieu of the transfer address code because the SYSTEM loader recovers the transfer address from the ISAM directory. Thus it needs to take action different from that when a standard load file has been completely loaded.

The next record types to discuss are those used in a generalized PDS file as exemplified in the PaDS/PRO-PaDS utility. Such a file starts with a record type X'06' in lieu of an X'05' which is the normal header type for a load module. This is used in certain utility commands to note whether the referenced file is a partitioned data set compatible with PRO-PaDS utilities.

The partitioned data sets include a MEMBER DIRECTORY which correlates the member NAME with its associated ISAM entry number. A representative PDS MEMBER DIRECTORY entry looks like this:

```
OC OB 64 69 72 20 20 20 20 01 01 7A OC ...
. . d i r . . z . ...
```

The TYPE=OC record indicates a PDS member directory entry record. The LENGTH byte specifies that the data area is an 11-byte field. The DATA AREA is subfielded as an 8-byte member name (stored in lower case), a one-byte ISAM entry number that is used to match up with a corresponding ISAM directory entry record, and a 2-byte field of member data. The first byte uses bit-7 to indicate a data member in contrast to an executable CMD program. Bit-6 indicates that the member has been established as "sector-origin" and can be directly accessed by linkage to the standard file access routines supported in PaDS/PRO-PaDS Version 2. Bit positions 5-4 are reserved for future use. Bits 3-0 and the next byte contain the 12-bit DATE field formatted as in the standard directory entry record. This entry is the date that the member was added to the PDS. The end of the MEMBER DIRECTORY is indicated by a TYPE=OE record with its expected length and data field (as in "OE 01 00"). The purpose of this record is similar to the TYPE=OA record for the ISAM directory. It indicates the end of the MEMBER directory. The ISAM directory is positioned in the PDS to follow the MEMBER directory.

One last set of record types to discuss is the records associated with the LDOS PATCH utility. When you apply an X-patch to a file, the name of the patch file is used as a header name with a record type of X'07'. Thus, if you want to YANK the patch, the PATCH program can read through the file and search for a like-named header. If a matching header is found, PATCH will change the header record type to a X'09' to indicate a yanked patch. Also, since it may be impossible to remove the patch without bubbling up any code blocks following the patch (another patch maybe?), PATCH will change the TYPE=01 records to TYPE=10 records. The TYPE=10 records will not be loaded by the SYSTEM loader but will be considered as non-loadable comment records.





Authored and copyrighted 1981, 1983 by Richard N. Deglin and Roy Soltoff. All rights reserved. PRO-CURE/CONV-CPM is published by MISOSYS, Alexandria, VA.

#### TABLE OF CONTENTS

General Information . . . . .	2
Distribution Diskette . . . . .	2
Supported CP/M Disk Formats . . . . .	3
The PRO-CURE/CONV-CPM Menu . . . . .	4
Command Details . . . . .	5
Running PRO-CURE/CONV-CPM From DOS Ready . . . . .	12
Error Messages . . . . .	15

Note: CP/M is a trademark of Digital Research, Inc.  
IBM is a trademark of International Business Machines Corp.  
LDOS is a trademark of Logical Systems Incorporated  
MAX-80 is a trademark of Lobo Systems, Inc.  
TRSDOS and TRS-80 are trademarks of Tandy Corp.

## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

### GENERAL

=====

PRO-CURE/CONV-CPM is a powerful tool which allows you to transfer files from various CP/M-formatted diskettes onto selected TRSDOS or LDOS-formatted disks. Nineteen different CP/M formats are directly supported under the PRO-CURE/CONV-CPM transfer utility. You can obtain CP/M disk directories, transfer files, and execute DOS commands easily and rapidly from the program's menu; these functions are also executable directly from DOS Ready.

### DISTRIBUTION DISKETTE

=====

This documentation covers the operation of both the LDOS 5.1 version (CONV-CPM) and the TRSDOS 6.x/LDOS 6.x version (PRO-CURE). The CONV-CPM utility is provided on a 35-track single density data diskette and is operational, under LDOS 5.1.x, on the TRS-80 Models I, III, and 4, and on the Lobo MAX-80. Certain CP/M formats may not be supported on the Lobo LX-80 Model I expansion interface [specifically 512-byte sector formats]. The PRO-CURE utility is provided on a 40-track single density data diskette and is operational under TRSDOS 6.x on the TRS-80 Model 4 only.



## SUPPORTED CP/M DISK FORMATS

=====

The following CP/M single-sided diskette formats are directly supported by PRO-CURE/CONV-CPM. No double-sided formats are readable. All formats are 5-inch mini-floppy, 35- or 40-track, unless specifically noted otherwise.

- 1) Cromemco Z-2 double-density
- 2) DEC VT-180 double-density
- 3) Heath/Zenith H89 single-density, soft-sectored
- 4) Heath/Zenith Z100 double-density
- 5) Holmes VID80 double-density
- 6) IBM Personal Computer CP/M-86 double-density
- 7) Kaypro II double-density
- 8) LNW80 double-density
- 9) Lobo MAX-80 5-inch double-density
- 10) Lobo MAX-80 8-inch double-density
- 11) Memory Merchant Shuffle Board double-density
- 12) Montezuma Micro Model 4, double-density, 256-byte sectors
- 13) Morrow Micro Decision double-density
- 14) NEC PC-8001A double-density
- 15) Omikron Mapper I single-density
- 16) Osborne One single-density
- 17) Standard IBM-3740, 8-inch single-density, CP/M format
- 18) Xerox 820-1 single-density
- 19) Xerox 820-2 double-density

## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

### THE PRO-CURE/CONV-CPM MENU

=====

The PRO-CURE/CONV-CPM utility is executed from DOS Ready by simply typing the following command:

CONVCPM<ENTER>                    [LDOS 5.1.x]  
or    PROCURE<ENTER>                [TRSDOS 6.x]

The PRO-CURE/CONV-CPM command menu will then be displayed:

```
CP/M disk type is not set
```

```
Control function : <S>et CP/M disk type  
                  <D>isplay CP/M disk directory  
                  <T>ransfer files from CP/M to DOS  
                  <C>ommand DOS  
                  <E>xit to DOS  
                  (S,D,T,C,E)? >
```

The first line of the menu indicates which CP/M format you have selected for operations. Initially, this type is not set.

Commands are entered on the prompt line. They may be typed, in either UPPER-case or lower-case, as any one of the single letters surrounded by brackets "<>", followed by <ENTER>. The <BREAK> and <BACKSPACE> keys may be used for correction as necessary.



# COMMAND DETAILS

=====

The following sections describe the function and operation of all PRO-CURE/CONV-CPM commands identified in the menu.

## <S>et CP/M disk type

-----

The <S>et command clears the screen and displays a menu of CP/M formats, any one of which may be selected for subsequent operations:

<S8> Standard CP/M; SSSD,8"	<OM> Omikron Mapper I; SSSD,5"
<X1> Xerox 820-1; SSSD	<X2> Xerox 820-2; SSDD
<L5> Lobo MAX-80; SSDD,5"	<L8> Lobo MAX-80; SSDD,8"
<KA> Kaypro II; SSDD	<OS> Osborne-1; SSSD
<VT> DEC VT-180; SSDD	<M4> Montezuma Model 4; DD,256
<PC> IBM PC CP/M-86; SSDD	<NE> NEC PC-8001A; SSDD
<H8> Heath H89; SD,soft-sector	<Z1> Zenith Z100; SSDD
<CR> Cromemco Z-2; SSDD	<LN> LNW80; SSDD,5"
<SH> MM Shuffle Board; SSDD	<HO> Holmes VID80; SSDD
<MO> Morrow MicroDecision; SSDD	

Enter type mnemonic? >

The term "SSSD" refers to single-sided single density media while the term "SSDD" refers to single-sided double-density media formats. Enter the desired diskette type by keying the appropriate two-letter mnemonic [the mnemonic is illustrated within angled brackets "<>"] followed by <ENTER>. You may use <BACKSPACE> to correct your entry. If you have typed a recognizable entry, you will be returned to the main menu, and the selected format will be noted at the top of the screen, above the menu. Any invalid entry will be ignored. Use the <BREAK> key to return to the main menu without changing the CP/M disk type.

<D>isplay CP/M disk directory

-----

The <D>isplay directory command reads the target CP/M diskette and then displays a directory of all or some of the files on that diskette, sorted by USER AREA and FILE NAME.

PRO-CURE/CONV-CPM will first prompt you to:

Enter CP/M drivespec or partspec? >

You may answer this prompt in one of three ways. You can simply key in the number of the drive in which the CP/M diskette is mounted, for example <:1>. Alternatively, you may enter a CP/M partial filespec, such as <ABC\*.ASM:1>. If you type <BREAK>, however, you will return immediately to the main menu.

A Note On CP/M Partial File Specifications

=====

A CP/M PARTSPEC consists of the following fields:

- 1) An optional FILE NAME of up to eight characters;
- 2) An optional FILE TYPE of up to three characters, preceded by a period <.>;
- 3) A required DRIVE NUMBER (1 through 7), preceded by a colon <:>.

The FILE NAME and FILE TYPE fields may contain the standard CP/M-style wildcard characters <?> and <\*>. The <?> character will match any other character in its position. The <\*> character must be the last character in the FILE NAME or FILE TYPE; it causes an automatic match for all positions in the FILE NAME or FILE TYPE including and following the <\*> character.

The FILE NAME or FILE TYPE, if not entered, will default to all spaces, unless both are omitted. In this case (i.e. when a CP/M DRIVESPEC only has been entered), they will default to <?????????.???>, which is equivalent to <\*. \*>.

The next prompt allows you to specify options in response to the prompt:

Enter parameters or <ENTER> for none? >

These optional parameters may be entered in a fashion similar to DOS command line parameters. For the <D>isplay directory command, you may select a



## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

particular CP/M diskette USER AREA, and you can obtain a copy of the CP/M directory on your printer device. Simply typing <ENTER> will set the parameters to their default values. As before, keying <BREAK> will return you to the main menu. The parameters are entered as follows:

USER=nn	Select a directory for USER AREA nn, where nn = 0 through 15. The default for this parameter is all USER AREAs. Abbrev: USER=U
PRINT=switch	Select a printed directory listing. Entering <ON>, <YES>, or <Y> for switch will cause the directory to be printed; entering <OFF>, <NO>, or <N> will suppress the printout. The default for this parameter is OFF. Abbrev: PRINT=P

At this point, PRO-CURE/CONV-CPM will read in the CP/M diskette's directory, sort it by USER AREA and FILE NAME/TYPE, and display all files which match the desired PARTSPEC and USER AREA. Each file is displayed in the following format:

u>nnnnnnnn.ttt sssK

where: "u" represents the USER AREA, in hexadecimal (0-9, A-F);  
"nnnnnnnn.ttt" represents the FILE NAME and TYPE; and  
"sssK" represents the FILE SIZE in kilobytes (x1024).

Up to three files will be shown on each display line. If the PRINT parameter is OFF, PRO-CURE/CONV-CPM will pause when the screen fills and wait for a keystroke; you may type <ENTER> to continue the display with the next screen, or <BREAK> to abort. If the PRINT parameter is ON, the directory display will scroll continuously without screen pauses.

If the CP/M diskette does not contain any files which match the desired PARTSPEC and USER AREA, the following message will appear:

No file(s) found!

In any case, when the directory display is complete, you will be prompted with the message:

Press any key to continue...

When you have done so, PRO-CURE/CONV-CPM will clear the screen and return you to the main menu.

## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

### <T>ransfer files from CP/M to DOS

-----

The <T>ransfer files command allows you to move files from your CP/M diskette to the DOS disk. This is the most powerful command in PRO-CURE/CONV-CPM.

The first prompt of this command is:

Enter source (CP/M) drivespec or partspec? >

This may be answered in the same way as the corresponding prompt in the <D>isplay directory command; you can enter a CP/M PARTSPEC, a CP/M DRIVESPEC, or <BREAK> to cancel the command.

The second prompt asks you to:

Enter destination (DOS) drivespec? >

Answer this prompt with the number of the drive onto which you want the CP/M files copied, such as <:0>. This must be an enabled DOS drive (hard disk or floppy). You may cancel the <T>ransfer files command at this point by keying <BREAK>.

Finally, you will be prompted:

Enter parameters or <ENTER> for none? >

Enter these optional parameters in a fashion similar to DOS command line parameters. For the <T>ransfer files command, you may select a particular CP/M diskette USER AREA, transfer all OLD or NEW files, or have PRO-CURE/CONV-CPM QUERY you for the transfer of each CP/M file. Simply typing <ENTER> will set the parameters to their default values. As described above, keying <BREAK> will return you to the main menu. The parameters are entered as follows:

USER=nn	Select transfers from USER AREA nn, where nn = 0 through 15. The default for this parameter is all USER AREAS. Abbrev: USER=U
OLD=switch	Transfer only files which already exist on the destination DOS disk. Entering <ON>, <YES>, or <Y> puts this qualification into effect, while <OFF>, <NO>, or <N> causes the checking for OLD files to be skipped. The default for this parameter is OFF. Abbrev: OLD=0
NEW=switch	Transfer only files which do not exist on the destination DOS disk. Entering <ON>, <YES>, or <Y> puts this qualification into effect, while <OFF>, <NO>, or <N> causes the checking for NEW files to be skipped. The default for

PRO-CURE/CONV-CPM Utility



# PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

this parameter is OFF. Abbrev: NEW=N

QUERY=switch

Query the user as to whether any given CP/M file is to be transferred. Entering <ON>, <YES>, or <Y> puts QUERY into effect, while <OFF>, <NO>, or <N> disables this option. The default for this parameter is ON. Abbrev: QUERY=Q

PRO-CURE/CONV-CPM will then read in the CP/M diskette's directory and sort it by USER AREA and FILE NAME/TYPE. Each file which matches the desired PARTSPEC and USER AREA, as entered above, then becomes a candidate for transfer. Next, PRO-CURE/CONV-CPM will check to see if the file exists on the destination DOS diskette. If the file does exist, and the OLD parameter is ON, PRO-CURE/CONV-CPM continues on to the next step. Similarly, if the file doesn't exist, and the NEW parameter is ON, the program continues. If neither case is met, the file is skipped.

At this point, if the QUERY parameter is ON, you will be asked:

Convert file u>nnnnnnnnn.ttt?

where "u" is the file's USER AREA and "nnnnnnnnn.ttt" is the file's NAME and TYPE. Answering <Y> directs PRO-CURE/CONV-CPM to then transfer the file. Responding <C> also directs PRO-CURE/CONV-CPM to transfer the file as well as turn the QUERY parameter OFF for all files remaining to be transferred. Any other response will cause the file to be skipped.

If the QUERY parameter is OFF, the following message will be displayed:

Converting file u>nnnnnnnnn.ttt

Since CP/M allows special (non-alphanumeric) characters in FILE NAMES and FILE TYPES, PRO-CURE/CONV-CPM must take exceptional action in such cases. If the QUERY parameter is ON, you will be prompted with:

Source CP/M filespec contains special characters  
Enter destination DOS filespec? >

Type in the name of a DOS file, without drive number, into which you wish the CP/M file's contents to be copied. Typing a <BREAK> at this point will cancel the <T>ransfer files command and return you to the main menu. If the QUERY parameter is OFF, the following question will be asked:

Source CP/M filespec contains special characters  
Convert this file?

An <N> answer will cause the file to be skipped, while a <Y> answer will proceed to the next prompt:

Enter destination DOS filespec? >

This is to be answered as described above.

## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

Finally, if the OLD parameter is OFF and the QUERY parameter is ON and the destination file already exists, you will be asked:

File exists -- replace it?

Typing <N> forces PRO-CURE/CONV-CPM to skip this file; typing <Y> will cause the existing DOS file to be overwritten with new data from the CP/M diskette.

Now PRO-CURE will read the CP/M file's contents into memory and then write the data out to the DOS disk. If the destination DOS disk becomes full during this process, you will be notified by:

Disk is full! - enter new output disk <ENTER>

Mount a different DOS diskette (one that has sufficient free space on it) in the destination drive and press <ENTER>. PRO-CURE/CONV-CPM then will continue its file transfer process. There are two cases, however, when a full destination disk will terminate the <T>ransfer files command: if the output drive is a fixed hard disk; or if PRO-CURE/CONV-CPM is executing as part of a DO command. The message:

Disk is full - can't continue!

will be displayed if this occurs.

The above sequence of events will be repeated for each CP/M file which matches the desired PARTSPEC and USER AREA, until no more are found. The following message will then be displayed:

File transfer complete

Alternatively, if no CP/M files existed on the source disk which matched these qualifications, then the following message will be displayed:

No file(s) found!

In any case, you will be prompted with the message:

Press any key to continue...

When you have done so, PRO-CURE/CONV-CPM will clear the screen and return you to the main menu.

Note that at any time during the file transfer process, pressing the <BREAK> key will abort the command after the current file has been completely transferred.



<C>ommand DOS

-----

The <C>ommand DOS function provides access to operating system commands from the menu level. Your DOS requests should be limited to library commands [a summary of library commands is normally obtainable via the <LIB> DOS command]. You will be prompted to:

Enter DOS command? >

After your command is keyed in, the screen will be cleared and your command line will be shown at the top of the video display. The DOS command will then be executed. At its completion, you will be prompted with the message:

Press any key to continue...

When you have done so, PRO-CURE/CONV-CPM will clear the screen and return you to the main menu.

<E>xit to DOS

-----

This command provides the means to terminate the PRO-CURE/CONV-CPM session and return to DOS Ready.





## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

The command line syntax for transferring files from a CP/M diskette to a DOS disk is:

=====	
<command> <CPM partspec> <DOS drive> (<type>,<parm>,<parm>,...)	
<command>	Type CONVCPM under LDOS 5.1.x Type PROCURE under TRSDOS 6.x
<CPM partspec>	This is REQUIRED and follows the rules given in the description of the <T>transfer files menu command.
<DOS drive>	This is REQUIRED and follows the rules given in the description of the <T>transfer files menu command.
<type>	This parameter is entered as any ONE of the CP/M formats listed in the CPM FORMAT TABLE, and is REQUIRED.
OLD=switch	Transfer only files which already exist on the destination DOS disk. Typing <ON>, <Y>, or <YES> for switch puts this qualification into effect while <OFF>, <N>, or <NO> causes the checking for OLD files to be skipped. The default for this parameter is OFF. Abbrev: OLD=0
NEW=switch	Transfer only files which do not exist on the destination DOS disk. Typing <ON>, <Y>, or <YES> for switch puts this qualification into effect while <OFF>, <N>, or <NO> causes NEW files to be skipped. The default for this parameter is OFF. Abbrev: NEW=N
QUERY=switch	Query the user as to whether any given CP/M file is to be transferred. Typing <ON>, <Y>, or <YES> puts QUERY into effect. Entering <OFF>, <N>, or <NO> disables this option. The default for this parameter is ON. Abbrev: QUERY=Q
USER=nn	Select transfers from USER AREA nn, where nn = 0 through 15. The default for this parameter is all USER AREAs. Abbrev: USER=U
=====	

In all respects, this command executes identically to the <T>transfer files command obtainable from the program menu, except that upon completion you are returned to DOS Ready.

# PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

The <type> parameter, as used in the above DOS command lines, is entered as a mnemonic [or abbreviation of the mnemonic] for one of the nineteen supported CP/M diskette formats as illustrated in the CPM FORMAT TABLE shown below.

CPM FORMAT TABLE		
Format	Type Parameter	Abbrev.
Cromemco Z-2	CROMEM	CR
DEC VT-180	VT180	VT
Heath/Zenith H89	HEATH	H8
Heath/Zenith Z100	Z100	Z1
Holmes VID80	HOLMES	HO
IBM Personal Computer CP/M-86	PC86	PC
Kaypro II	KAYPRO	KA
LNW80	LNW80	LN
Lobo MAX-80 5-inch	LOB05	L5
Lobo MAX-80 8-inch	LOB08	L8
Memory Merchant Shuffle Board	SHUFFL	SH
Montezuma Micro Model 4	MONTEZ	M4
Morrow Micro Decision	MORROW	MO
NEC PC-8001A	NECPC	NE
Omikron Mapper I	OMIKRN	OM
Osborne One	OSBORN	OS
Standard 8-inch CP/M format	STD8	S8
Xerox 820-1	XEROX1	X1
Xerox 820-2	XEROX2	X2



## ERROR MESSAGES

=====

Any one of the following error messages may be displayed:

### Invalid CP/M filespec, or missing CP/M drivespec!

An incorrect CP/M PARTSPEC or DRIVESPEC was entered in a DOS command line, or in response to a prompt requesting same.

### Invalid or missing DOS drivespec!

An incorrect DOS DRIVESPEC was entered in a DOS command line, or in response to a prompt requesting same.

### CP/M drive can't be :0!

You cannot read a CP/M diskette in drive 0.

### CP/M and DOS can't be same drive!

A file transfer command requires the CP/M and DOS drives to be different.

### Invalid or missing CP/M disk type!

One, and only one, of the acceptable CP/M format parameters was not entered in a DOS command line requesting a file transfer.

### Parameter error!

An invalid parameter keyword was entered, or an ON/OFF-type parameter was given an illegal value.

### Drive is not 5-inch! or Drive is not 8-inch!

The CP/M drive selected was not of the correct size, given the desired CP/M format.

### Not a floppy drive!

The CP/M drive selected was not a 5-inch or 8-inch floppy.

### Not a double-density controller!

The floppy controller in your system cannot support the double-density CP/M format selected.

### Invalid CP/M user area!

The USER parameter was entered with a value not in the range <0 to 15>.

## PRO-CURE/CONV-CPM - CP/M to DOS File Transfer Utility

### Insufficient room for command execution!

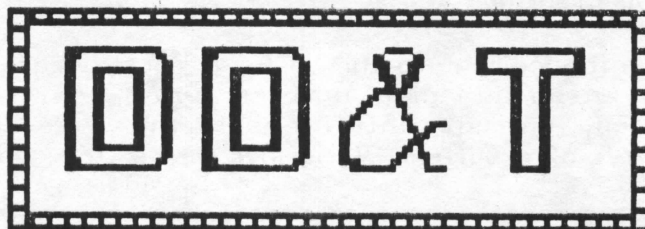
You attempted to execute a DOS command from the program menu, but there is not enough system memory available to do this.

### System error messages

Various DOS system error messages may also be displayed; please see your DOS manual for an explanation if such occurs.



## DD&T - Debugger Disassembler and Trace Utility



### TABLE OF CONTENTS

General Information . . . . .	1
DD/CMD - Debugger Disassembler . . . . .	2
DOS Program Trace Utility . . . . .	3
PTRACE - Program Trace Module . . . . .	3
STRACE - Statistical Trace Report . . . . .	4

DD/CMD, PTRACE/CMD, and STRACE/CMD: Copyright 1984 by Richard N. Deglin, All rights reserved. DD&T is published by MISOSYS, Inc., Sterling VA 22170.

LDOS and LS-DOS are trademarks of Logical Systems Inc.  
TRSDOS is a trademark of Tandy Corp.

### GENERAL INFORMATION

This documentation covers the Model I/III version of DD&T which functions under the LDOS 5.1 operating system. It also covers the TRSDOS 6.x or LS-DOS 6.x version of DD&T called PRO-DD&T. The specific version of DD&T is noted on the diskette label supplied with this package. The DD&T package provides the assembler programmer with a major enhancement to the debugger supplied with your DOS. The Debugger Disassembler module is completely relocatable. Its interface to the DOS debugger is automatic. The TRACE utility can be used to help hone your programs to optimum efficiency. DD and TRACE together make DD&T a set of fine tools crafted to provide you with a rewarding assembler programming experience.

## DD&T - Debugger Disassembler and Trace Utility

### DD/CMD - Debugger Disassembler

DD/CMD is an enhancement to the DOS system debugger which provides the added capability of an on-line disassembler during your program debugging sessions. Once invoked, it resides interfaced to the system's DEBUG module - available at the touch of a button. DD is invoked via the command:

```
=====
|
|   DD (parameter)
|
|   ON      Loads the DD module into high memory
|           and links into the system debugger.
|
|   OFF     Unlinks DD from the debugger and
|           reclaims high memory if possible.
|
|   Abbr: ON=YES or Y, OFF=NO or N
|
|=====
```

The command "DD (ON)" or "DD" loads the DD high memory module and links itself into the system debugger. DD may be SYSGENed if you find it convenient to have the Debugger Disassembler available at each boot. The high memory module takes less than 1600 bytes of memory. A subsequent entry into the system debugger (normal or extended) activates the DD module. In the upper right hand corner of the debugger register display ("X" mode), DD will display a mnemonic disassembly of the Z80 instruction which the current program counter (PC) points to. After a single-stepping debugger command ("I", "C", or "J") is executed, or a breakpoint is reached via use of the "G,nnnn" command, DD will update this instruction disassembly automatically.

DD also provides for Z80 disassembly directly from any memory location in your computer's 64 Kilobyte address space. The command "Z<ENTER>" will start the disassembly from the current program counter. The command "Znnnn<ENTER>" will start the disassembly from the hexadecimal address given in the command as "nnnn". The screen will clear, and 16 or 24 lines of disassembled instructions will be displayed, depending on whether you are operating version 5 or 6 of the DOS. You may continue the display with one or more additional screenfulls of disassembly by typing any keystroke; if you type "X", however, you will be returned to the prior debugger display screen ("X" or "S" mode).

The DOS command "DD (OFF)" will unlink DD from the system debugger and attempt to reclaim high memory space. If DD is the first module found in protected high memory, the space will be reclaimed and the protected memory pointer (HIGH\$) adjusted. If the DD module is not the first protected module, DD will unlink but the high memory allocation will remain. If the latter is the case, and you reload DD with the ON parameter, it will reuse the same high memory space as it previously occupied.

## DD&T - Debugger Disassembler and Trace Utility

### DOS Program Trace Utility

The function of the TRACE utility is to simply show you where your program spends most of its time in execution. Programs may be coded so that they waste too much time in inefficiently written program routines. TRACE finds these program sections for you by means of the Statistical TRACE Report. You should compare your program listing against the TRACE report and analyze the routines where your program spends its time. Start tracing with a coarse address range. Once you identify major sections, you can narrow the trace to produce a more detailed snapshot of a given piece of code. Ask yourself if the section of code should normally be executed more frequently than other sections. If there is no reason for such frequent execution, look for another way to code the routine. By reducing these bottlenecks of inefficient routines, your programs will run faster.

The DOS program trace utility is a package of two utilities designed to help you optimize the design and coding of an assembler application program. It consists of two utilities; PTRACE which records program activity by execution address ranges and, STRACE which compiles and displays the statistics of a series of PTRACE runs.

### PTRACE - Program Trace module

PTRACE allows you to execute the target program in an environment which maintains a record of all program activity. Note that you cannot directly trace a library member. Your selected address range is divided evenly into 256 "buckets". PTRACE keeps a counter for each bucket. Two other buckets are used to track activity in areas above and below the selected range. Every time a system heartbeat interrupt occurs, PTRACE determines the program counter at the time of interrupt and updates the counter corresponding to the correct bucket. PTRACE is invoked via the command:

```
=====
PTRACE trace-file (START=X'nnnn',END=X'nnnn')
```

```
trace-file The filespec to receive the trace
            data table generated during the
            target program's invocation.
```

```
START      The lower address range to obtain
            trace data.
```

```
END        The upper address range to obtain
            trace data.
```

```
Abbr: START=S, END=E
=====
```



## DD&T - Debugger Disassembler and Trace Utility

The trace file is written out to disk when the trace terminates; i.e. when the target program exits to DOS Ready. START and END determine the address range; they must be greater than 255 bytes apart. The defaults are START=X'0000' and END=X'FFFF'. To keep interrupt overhead to a minimum, the range is rounded up to the next higher power-of-2; e.g 256, 512, 1024. The maximum range is all of memory, or 65536. If the table filespec [trace-file] is not entered on the DOS command line, PTRACE will prompt for it. In any case, PTRACE will prompt for the command line which will execute the target program. This is entered identically to the DOS command line which would be used to invoke the target program normally. PTRACE will load and execute the target program, trace its activity, and terminate by saving the bucket counters and associated range data as a Program Trace Table to a disk file, with default extension /PTT. While the trace is active, PTRACE will place a blinking asterisk (\*) in the upper right corner of the video screen.

Ten sample trace tables are included on this disk as T0/PTT through T9/PTT.

### STRACE - Statistical TRACE Report

STRACE will report statistics computed from the data in one or more trace result files. In many cases, it will be desirable to repeat a PTRACE run several times. This depends on the range of the trace, how long the target program executes, and how fast your DOS computer is running (2 MHz or 4MHz, for instance). If a range is wide, the trace results will be coarser. If a target program run is short, fewer statistics will be gathered. If your computer is running fast, the fixed system interrupt interval will fall behind program activity. In any case, the more trace runs you make, the better the statistics will be. Use of a Job Control Language file to invoke PTRACE will ease this process.

STRACE is invoked by the command:

**STRACE [trace-file-1] [trace-file-2] ... [trace-file-N]**

The file extension default of "/PTT" will be added to the file specification if you omit it. The report can be redirected to the printer by appending ">\*PR" to the command line, or to a file by appending ">report file" to the command line. The quotes are not entered. At least one trace file specification must be entered, but their order is not significant if more than one is present.

If a consecutive series of "buckets" within the trace range all have a count of zero occurrences, they are squeezed together to form one "bucket" in the output listing.

You cannot generate an STRACE report from trace tables with different address ranges. A sample trace report is included on this disk as TEST/RPT.

## DESCRIBE - Directory Descriptor Extension

### TABLE OF CONTENTS

General Information . . . . .	1
Invoking DESCRIBE . . . . .	2
Alter a descriptor field . . . . .	3
Change to a new drive . . . . .	3
DOS Command request . . . . .	4
File directory display . . . . .	x
Help information . . . . .	x
Interchange data LOAD and SAVE . . . . .	x
Modify listing format . . . . .	x
Remove descriptors from a disk . . . . .	x
Search for a string . . . . .	x

DESCRIBE: Copyright 1984 by MISOSYS, INC., All rights reserved. DESCRIBE is published by MISOSYS, Inc., Sterling VA 22170.

LDOS and LS-DOS are trademarks of Logical Systems Inc.  
TRSDOS is a trademark of Tandy Corp.

### GENERAL INFORMATION

This documentation covers the Model I/III version of DESCRIBE which functions under the LDOS 5.1 operating system. It also covers the TRSDOS 6.x or LS-DOS 6.x version of DESCRIBE called PRO-DESCRIBE. The specific version of DESCRIBE is noted on the diskette label supplied with this package. The program and its overlays are stored in a Partitioned Data Set. This file has a password which limits its access to READ. If for any reason you need to write to the DESCRIBE/CMD file (such as to apply a patch) or to RENAME the file, or any other access which is greater than READ, use the password, ".DESCRIBE".

DESCRIBE provides the user with an extension to the DOS directory which incorporates a 63-character descriptor record. This facility gives you the convenience of a complete description for each file - anytime that the file is accessible.

## DESCRIBE - Directory Descriptor Extension

### INVOKING DESCRIBE

DESCRIBE is easily invoked in two formats. To invoke the utility for a maintenance purpose (such as to edit a descriptor field), just enter the command,

#### DESCRIBE

or whatever name you have RENAMED the program data set. A menu of operations will be displayed. If you only want to display a file directory display for a disk that has been extended with descriptors, invoke DESCRIBE with the command,

#### DESCRIBE ambigspec

where "ambigspec" is considered to be an ambiguous file specification. This takes the form of a file name field, a file extension field, and a mandatory drive specification. Within the filename and extension fields, the character "\*" will match all other characters remaining in the field while the character "?" will match all other characters in that position. If the filename field is blank, it will default to "\*". If the extension field is blank, it will match only a file with no extension. If you omit both fields, the ambigspec defaults to \*/\*:d.

If you select the maintenance mode of invocation, the displayed menu will look something like the following screen.

DESCRIBE 1.0 Copyright (c) 1984 MISOSYS, Inc.	
<A>lter descriptor field	<I>nterchange data
<C>hange to a new drive	<M>odify listing format
<D>OS Command request	<R>emove descriptors
<F>ile directory display	<S>earch for a string
<H>elp information	e<X>it to DOS

DESCRIBE is a tool to extend your disk directory with a descriptor field for each file. The field is 63 characters in length and is used by you to add information describing each file stored in the directory. DESCRIBE provides commands to manage these descriptors as well as provide you the means to construct customized sorted directory displays to the display screen, your printer, and even a disk file. DESCRIBE also has a command to allow you to invoke a DOS command as if you were at DOS Ready. While the main menu is displayed, a blinking cursor is positioned to the left of a command letter. Commands can be accessed by depressing the letter contained within angle



## DESCRIBE - Directory Descriptor Extension

brackets or by moving the cursor via UP/DOWN arrows to the command "word" then depressing <ENTER>.

### Alter a Descriptor

A is used to ALTER or EDIT the descriptor field. When you enter this command, the screen will display a file's name along with the MOD date and attributes. If any descriptor field is present, it will also be displayed. The prompt tells you to use the UP/DOWN/ENTER keys to scroll through each filespec. If you wish to edit a descriptor field, depress the letter "E". The editing keys at your disposal are:

CLEAR-LEFT-ARROW	- Delete the character at the cursor and shrink the line by one position
CLEAR-RIGHT-ARROW	- Expand the line by one character
LEFT-ARROW	- Move the cursor left one position
RIGHT-ARROW	- Move the cursor right one position
ENTER	- Store changes and proceed to next file
BREAK	- Abort changes and proceed to next file

If the file's name is prefixed with a question mark, it means that file has been added to the disk since the directory was extended. This "flag" will automatically disappear once you edit the descriptor field. As an aside, once you delete a file from a directory after you have "described" it, the description will not come up during the ALTER command.

### Change to a New Drive

During the maintenance operations, the directory descriptor extension for a diskette is kept in a memory buffer. The CHANGE command is used to identify the disk drive you want to describe. You must use this command to identify the first disk to operate with. Thereafter, each time you wish to change to a different disk, YOU MUST INVOKE THIS <C>HANGE COMMAND PRIOR TO SWITCHING DISKETTES. That's to give DESCRIBE an opportunity to update the descriptor data and close the directory. The program attempts to ensure that the disk to update is the same as the current one by matching the disk pack identification. This is not failsafe! It is strongly recommended that you assign a unique disk name to each diskette in your collection. Aside from the importance for a program such as DESCRIBE to uniquely identify each diskette from another, catalog programs such as ZCAT need uniquely assigned diskette names. Its a good practice to get into.

### DOS Command Request

This DESCRIBE command allows you to enter any command acceptable at the DOS Ready prompt. Your command will be invoked by the DOS. At its conclusion, a prompt to "Press <ENTER> to continue" will be displayed. After you depress the <ENTER> key, the DESCRIBE menu will re-appear; the descriptor buffer will

## DESCRIBE - Directory Descriptor Extension

"break" command and all systems will continue to operate as if the command had not been entered.

### File Directory Display

This "F" command is used to obtain the directory listing of files. The display is formatted according to the user defined format specification. This specification can be customized via the "M" command. The listing will be titled if the first character of the specification is a plus sign ("+" ). The title is constructed with headings appropriate to each data item in the listing (see the "M" command for titling details). A line of dashes completes the title.

The listing format for each file is controlled by the sequence of keywords and other characters contained in the format specification. When you invoke the "F" command, the prompt,

Output to <D>isplay, <P>rinter, or <F>ile?

will be displayed in the status line. This gives you an opportunity to select the destination device for the listing. Depressing "D" followed by <ENTER> will select the video display screen. This will produce a display paged according to your screen size. Depressing "P" followed by <ENTER> will produce a printer listing. If you select the "File" output, you will be prompted to enter the name of the disk file to which you want the output written. Depressing <ENTER> by itself will also select the video display screen paged listing. A <BREAK> will abort the operation.

### Help Information

This command provides eight screens of text which describe various functions of the DESCRIBE program. The help screens are useful once you become slightly familiar with the DESCRIBE facility by providing you with an abbreviated on-line user manual.

### Interchange Data

This command allows you to load or save the directory data identified by the format specification from/to a data file. The file is structured in the Data Interchange Format (DIF). The load operation will extract the descriptor fields from the DIF file loaded and update the current set of descriptors when the file specification matches. The load operation requires that the following fields are present in the DIF file: \$DES and \$SPC or \$DES and \$NAM (or \$NAX) and \$EXT (or \$EXX). An error will be generated if the required fields are not present. The save operation creates a DIF file in column format.

A typical use for the generation of a DIF file would be to transfer the contents of the descriptor fields to a backup disk. A non-typical use would

## DESCRIBE - Directory Descriptor Extension

be to create a DIF file of directory information and load the data into a spreadsheet program - although its quite possible.

### Modify Listing Format

The MODIFY command allows you to alter the format specification to customize the file directory display listing. When you invoke "M", the current format will be displayed along with a listing of the keywords supported. This modifies the menu screen with an addition such as the following box. The format specification illustrated is the default format generated by DESCRIBE when a descriptor extension is first created.

```
ATT DAT DES DEX DRV EOF ERN EXT EXX LRL  
NAM NAX PRO REC VID VNM VDT
```

```
+$spc $att $pro $lrl $rec $eof $dat;$des;)
```

```
<ENTER> to save edits, <BREAK> to abort edits
```

The two rows of keywords are displayed as memory joggers. The keys should be quite descriptive for the values they stand for. The Format Table provides the keyword description and the title string generated when a title is requested. Within the format, each keyword must be prefixed with a dollar sign. If the first character is set to a "+", the listing will include a title. The string must be terminated with a closing parenthesis. If you omit it, a closing parenthesis will automatically be applied when you save the format. The semicolon, ";", can be used to specify a logical carriage return. This should be used if your format contains a line which is going to exceed the screen width. The format string is edited with the same edit functions as the ALTER command. DESCRIBE provides a maximum of 64 characters for the format string.



## DESCRIBE - Directory Descriptor Extension

----- Format Table -----	
Key Definition	Title String
ATT File attributes; "+*SIPC"	Attrib
DAT Mod date; "dd-mmm-yy"	Mod date
DES Descriptor field	File Description
DEX Descriptor field to 63 chars	File Description
DRV Disk drive; ":d"	D_
EOF End-of-file offset; "ddd"	Eof
ERN Ending record number; "ddddd"	_Ern_
EXT File extension	Ext_
EXX File extension to 3 chars	Ext
LRL Logical record length; "ddd"	Lrl
NAM File name	Filename
NAX File name to 8 chars	Filename
PRO Protection level; "prot"	Prot
REC Number of records by LRL; "ddddd"	Nrec_
SPC File name and extension string to 12 chars	Filespec_____
VDT Volume date; MM/DD/YY	Vol Date_____
VID Volume name and date to 16 chars	Disk Pack ID_____
VNM Volume name to 8 chars	Vol Name

### Remove Descriptors

The REMOVE command allows you to delete the descriptors from the directory of the currently logged disk. This facility is the only way you can properly restore the directory to its standard size. You will be prompted to continue the removal operation prior to its execution. This is a safeguard so you don't inadvertently delete all of the descriptors that you so painstakingly edited. If you want to remove the descriptors but restore them at a later date, why not first prepare an interchange file with \$SPC and \$DES data so that the descriptor contents can be saved for the restoral?

### Search for a String

The SEARCH command allows you to invoke a file directory listing which includes all files with a descriptor character string that matches your search string. Your search string can be up to 32 characters in length. The matching is performed without regard to UPPER/lower case (it is case insensitive). By using this command, you can obtain a directory display based on a file's description rather than on just its name.

## PRO-ESP - Enhanced System Package

### TABLE OF CONTENTS

ALTDISK . . . . .	2
ALTLD . . . . .	4
ALTRES . . . . .	5
CRLF/FLT . . . . .	6
CTLG/FLT . . . . .	7
CVT324 . . . . .	8
DED . . . . .	9
DOEDIT/FLT . . . . .	14
FKEY . . . . .	16
IOMON . . . . .	17
MINIDOS/FLT . . . . .	20
NAME . . . . .	21
PRTOGGLE . . . . .	22
RD40 . . . . .	23
UNREMOVE . . . . .	25
XONXOFF/FLT . . . . .	26

All programs: Copyright 1984 by Richard N. Deglin, Riclin Computer Products and Karl A. Hessinger, MicroConsultants, All rights reserved.  
PRO-ESP is published by MISOSYS, Inc., Sterling VA 22170.

LS-DOS is a trademark of Logical Systems Inc.  
TRSDOS is a trademark of Tandy Corp.

### General Information

This documentation covers the TRSDOS 6.x or LS-DOS 6.x (herinafter referred to as DOS) version called PRO-ESP. The PRO-ESP package provides the user with a collection of valuable utility programs and filters designed to enhance the operation of your DOS. The package of utilities is provided on a 40-track double density formatted data diskette.

## PRO-ESP - Enhanced System Package

### ALTDISK/CMD

ALTDISK is a disk-drive simulator which creates a 32K or 64K DOS drive in the second (64K) bank of RAM. ALTDISK uses less than 30 bytes of low memory and about 100 bytes of high memory, and can be operated with all DOS products, including a hard disk driver or the I/O Monitor. You can load the DOS system files into the RAM disk, and use it as drive 0 for very fast system response. ALTDISK can be operated as any one of the eight allowable DOS drives, and upon invocation the target drive can be formatted, or re-enabled if previously formatted.

ALTDISK (parm, parm, ...)

DRIVE=n            Specify the drive (1-7) which will  
                    be the RAM disk. Defaults to drive 7.

FORMAT=sw        ON, Y, or YES will format the drive,  
                    and erase all previous information.  
                    OFF, N, or NO will reenable a  
                    previously formatted RAM drive, with  
                    all information intact. Default in ON.

HALF=sw           ON, Y, or YES will cause ALTDISK to  
                    use only the 1st bank of alternate  
                    RAM, giving a 32K drive.  
                    Defaults to OFF.

OFF               Will disable ALTDISK and reclaim  
                    high memory if possible.

abbr: DRIVE=D, FORMAT=F, HALF=H, OFF=N or NO

ALTDISK will create or re-enable a RAM disk which, in its full configuration, has 16 cylinders, 4 granules per cylinder, and 4 sectors per granule. The files BOOT/SYS and DIR/SYS will be on the drive. BOOT/SYS uses one granule of cylinder 0, while DIR/SYS uses all of cylinder 1. This gives you 59K of free space, with directory capacity for 112 files. A half-size drive has 8 cylinders containing 27K of free space.

When ALTDISK is invoked, it will inform you, if all goes well, that the selected drive has either been created or re-enabled:

**Drive n formatted and enabled (if FORMAT=ON)**

**Drive n reenabled (if FORMAT=OFF)**

If, when ALTDISK is invoked, the upper alternate bank is currently in use, then ALTDISK will create a half size (32K) memory disk and will display an appropriate message.



## PRO-ESP - Enhanced System Package

When you remove ALTDISK, one of the following messages will be displayed:

**ALTDISK removed, high memory not reclaimed**  
**ALTDISK removed, high memory reclaimed**

To use ALTDISK as drive 0, first invoke it as some other drive, backup all system files to it (except SYS0/SYS), and enter the SYSTEM (SYSTEM=n) command to switch the RAM disk to drive 0. For example:

**ALTDISK (DRIVE=7,HALF=NO,FORMAT=YES)**  
**BACKUP SYS/SYS:0 :7 (SYS)**  
**REMOVE SYS0/SYS.LSIDOS:7**  
**SYSTEM (SYSTEM=7)**

If an error occurs when ALTDISK is invoked, one of the following messages will be displayed:

**Parameter error!** - An invalid parameter was entered on the command line. Reenter the command line with correct syntax.

**Requested drive already enabled!** - The requested drive already exists in your system; it may be any type of drive (floppy, hard, RAM disk). Use a drive which is currently inactive.

**Invalid drive number!** - A drive number not in the range of 1 to 7 was entered.

**ALTDISK already active, drive n!** - An ALTDISK RAM drive exists in your system, but it wasn't the requested drive. More than one RAM disk cannot be active at the same time. To change the drive number of an existing RAM disk, reboot and invoke ALTDISK with the new drive number and FORMAT=OFF.

**Can't - only valid at DOS Ready** - You attempted to invoke ALTDISK from BASIC or a similar situation. Since ALTDISK alters the system high memory pointer, it can only be invoked from DOS Ready.

**Lower alternate bank already in use, ALTDISK not installed!** - You attempted to install ALTDISK when the lower alternate bank was already in use.

**Can't remove drive 0!** - You attempted to remove ALTDISK when installed as drive 0.

**ALTDISK not installed!** - You attempted to remove ALTDISK when it was not active.

**Drive code table doesn't point to ALTDISK, can't remove!** - ALTDISK has determined that it has been "filtered"; remove the "filter" first.

NOTE: ALTDISK cannot be SYSGENed.

## PRO-ESP - Enhanced System Package

### ALTLD/CMD

The ALTLD utility provides a simple way to rapidly save and restore the entire contents of the alternate RAM banks to and from a disk file.

ALTLD filespec/RAM (parm)	
LOAD	Loads the contents of the specified file into the alternate banks.
DUMP	Saves contents of alternate banks to a disk file.
abbr: DUMP=D, LOAD=L	

If the DUMP parameter is specified, ALTLD will dump the entire contents of the alternate 64K of RAM to a disk file. This file may later be loaded back into the alternate RAM by use of the LOAD parameter.

If the LOAD parameter is specified, ALTLD will load the contents of a previously dumped file into the alternate RAM banks. ALTLD will check the length of the specified file, and will only load the file into alternate RAM if the file is exactly 64K bytes in length; if this is not the case, an appropriate error message will be displayed and ALTLD will abort. This safeguard is there if you inadvertently attempt to load a file which is not a RAM image.

If no parameters are specified, you will be prompted first for the RAM filespec and then:

**<L>oad or <D>ump?**

Type <L> to load RAM from the file or type <D> to dump RAM into the file.

If you specify BOTH the DUMP and the LOAD parameters on a command line, the following message will be displayed and ALTLD will return to DOS Ready:

**Can't dump AND load!**

While ALTLD is loading or dumping the contents of the alternate RAM, it will display an incrementing memory page counter (in hexadecimal) as an indication of its progress. Upon finishing the process, a completion message will be displayed and you will be returned to DOS Ready.

## PRO-ESP - Enhanced System Package

### ALTRES/CMD

ALTRES is a replacement for the SYSTEM (SYSRES=nn) command which places a specified system overlay into the upper alternate bank of memory.

ALTRES (parm, parm...)

SYSRES=nn      Load system overlay nn into alternate RAM.

OFF            Disable ALTRES and reclaim memory if possible.

abbr: SYSRES=S, OFF=N or NO

ALTRES is a replacement for the SYSTEM (SYSRES=nn) command. ALTRES functions in the same way as SYSTEM (SYSRES=nn) except that the system modules are stored in the upper alternate bank. ALTRES only uses the upper of the two alternate banks, so if you wish to also use the ALTDISK driver, specify the HALF parameter to instruct ALTDISK to only use the lower bank. ALTRES will display an error message and abort if the upper alternate bank is already in use. ALTRES will also abort if the SYSTEM (SYSRES) high memory hook is already active.

Specifying the OFF parameter will instruct ALTRES to remove itself from the system; and if possible, reclaim the space it used in high and low memory. Specifying the OFF parameter when ALTRES is not active will cause the error message:

**ALTRES not installed!**

to be displayed.

ALTRES may be SYSGENed, with the following in mind. When ALTRES is initialized via the system boot configuration (@ICNFG) vector at reset time, it will perform a checksum on the alternate bank to determine if the system modules which were ALTRESed are still resident. This will allow ALTRES to recover from a reboot cleanly. If the checksums for any of the previously resident system modules are in error, then the ALTRES high memory hook will be active, but those particular modules will not be resident. Those modules which checksummed correctly will be resident again. In any case, you can invoke a JCL (using ALTRES) which will reload any or all of the desired system overlays back into the alternate bank.

NOTE: SYS0, SYS6, SYS7, SYS8, and SYS13 (if present) cannot be made resident.



## PRO-ESP - Enhanced System Package

### CRLF/FLT

The CRLF filter allows the DOS video driver to properly handle carriage return/linefeed pairs sent to it.

Set \*devspec [to] CRLF/FLT  
Filter \*DO [using] \*devspec

There are no parameters. [Use CR for devspec]

This filter is to be installed on the \*DO device whenever you wish to properly handle carriage return/linefeed pairs sent to the video. This is accomplished by changing every carriage return control character (X'0D') sent to the filtered device into a beginning of line character (X'1D'); linefeed characters (X'0A') are passed to the device unchanged. This filter will find use in conjunction with communications programs, such as COMM/CMD, whenever you are communicating with a remote computer system which transmits CR/LF pairs at the end of each line. Many mainframe computer systems use this method to terminate lines of output; without the use of the CRLF/FLT, reception of a CR/LF pair would cause double-spacing on your video display. Note that when using COMM with this filter, you must turn on the COMM option "Accept Linefeed" (<CLEAR-SHIFT><4>, <CLEAR><:>).

## PRO-ESP - Enhanced System Package

### CTLG/FLT

The CTLG filter will cause your machine to beep every time an ASCII BEL character is sent to the filtered output device.

Set \*devspec1 [to] CTLG/FLT  
Filter \*devspec2 [using] \*devspec1

There are no parameters. [Use CG for devspec1]

This filter will produce an audible beep using your machine's internal speaker whenever an ASCII BEL character (X'07') is sent to the filtered output device. Normally, the filter is installed on the \*DO device; however, any output device will do. This filter will find use in conjunction with applications which require the generation of an audible tone.

To demonstrate the use of this filter, install it on the \*DO device, and, using COMM/CMD, type <CTRL><G> with the half duplex option on. Every time <CTRL><G> is depressed, you will hear a beep.

For convenience, the constants used to generate the beep sound are located at X'3000' (tone) and X'3001' (duration) in the CTLG/FLT load module. These are passed to the system @SOUND SVC (the tone in bits 0-2 and the duration in bits 3-7 of register B) by the filter. The PATCH utility may be used to easily change these values, and thus the Control-G sound, if your sound generation hardware supports a variety of tones and/or durations.

## PRO-ESP - Enhanced System Package

### CVT324/CMD

CVT324 will convert your DOS Version 5 BASIC programs to run on the DOS Version 6 BASIC.

```
CVT324 input_file output_file
```

There are no parameters.

CVT324 will help you convert your old BASIC programs to run on DOS Version 6 BASIC. The input file must be a BASIC program stored in compressed format. The output file will be an ASCII file which can be loaded into BASIC. CVT324 will perform the following conversions on your program:

- 1) Add spaces around keywords.
- 2) Strip any trailing information after a CLEAR statement.
- 3) Convert PRINT @ statements for 80 x 24 screen.

Besides the conversions, any line containing one of the keywords; INPUT, OUTPUT, POKE, PEEK, or USR, will cause the message:

**The line above contains a questionable statement**

to be displayed. Because of the differences between the BASIC provided with DOS Versions 5 and 6, these keywords will probably require changes for the program to run properly under DOS 6.

Any line containing one of the keywords; SET, RESET, POINT, CLOAD, CSAVE, or SYSTEM will cause the message:

**The line above contains a bad token and cannot be converted**

to be displayed. The above keywords are not supported by the BASIC supplied with DOS Version 6 and must be changed before the program can be used under this version of BASIC.



## PRO-ESP - Enhanced System Package

### DED/CMD

The Disk Editor allows you to easily modify the contents of any DOS compatibly formatted disk.

DED :d

There are no parameters.

If the drive number is not specified on the command line then you will be prompted to enter it. Once a drive has been selected, DED will scan the drive and then display the first sector on the disk in the following format:

0123456789ABCDEF	Byte	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
..&!.....&\$..	<00>	00	1E	26	21	00	10	01	0F	03	CD	BB	04	26	24	01	01
...&.....!p.:	<10>	01	CD	BB	04	26	14	04	0C	CD	BB	04	FD	21	70	14	3A
...o.Ad.I. ....E	<20>	D8	07	CB	6F	01	41	64	11	49	1D	20	05	CB	A8	11	45
..w ....o.' ..\$	<30>	11	CB	77	20	0C	CB	80	CB	6F	11	27	0F	20	03	11	24
.....w..q..r.	<40>	09	FD	7E	03	E6	03	80	FD	77	03	FD	71	04	FD	72	07
.s.:...w. _.&#..	<50>	FD	73	08	3A	02	04	FD	77	09	5F	06	00	26	23	CD	BF
..K.#x. ....w.>"	<60>	04	ED	4B	CC	23	78	E6	20	FD	B6	04	FD	77	04	3E	22
..w.%.....: #..2	<70>	81	FD	77	06	25	06	04	CD	BF	04	3A	00	23	E6	10	32
.\$!....>..2...2.	<80>	0F	24	21	00	13	06	10	3E	10	90	32	E0	07	7E	32	E1
..#..>.!..U..>...	<90>	07	23	10	F3	3E	04	21	00	15	55	5D	F5	3E	09	F1	01
.....>.....>....	<A0>	00	04	CD	D6	04	3E	06	CD	D6	04	0B	3E	02	CD	CE	04
...>.....\$....H	<B0>	11	00	0C	3E	01	CD	CE	04	C3	00	24	1E	00	18	01	48
.W...!.....	<C0>	AF	57	FD	E5	FD	21	D8	07	CD	03	00	FD	E1	C9	D5	E1
.2..6 ..2.....	<D0>	13	32	DC	07	36	20	D5	C5	32	DC	07	ED	80	C1	D1	C9
.....	<E0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
.....	<F0>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Drive : 1 Cylinder : 0 Sector : 0 Byte : X'00' = X'00' = 00000000 = 0  
Sector assigned to : BOOT/SYS

Command :

The largest portion of the screen is taken up by the display of the sector buffer. The ASCII characters on the left correspond with the hexadecimal values to the right. If a byte has a value less than X'20' or greater than X'7F', it will be displayed in the ASCII section as a period.

Beneath the sector display are two lines of information. The first informs you of your current position on the disk. This line contains the current drive, cylinder, sector and the relative byte position within the sector. Following the relative byte position, the value of the byte at that position is displayed in hexadecimal, binary, and decimal formats.

## PRO-ESP - Enhanced System Package

The second information line contains the name of the file to which the current sector is assigned. When DED logs in a disk, it creates a map of the disk's file/sector assignments, which are then displayed on this line. DED does not support the ability to position to a relative sector within a file; it will only inform you as to which file the displayed sector is assigned.

Before looking at the list of commands that DED supports, it is important to make note of two items. First, any command may be aborted at any time by depressing the <BREAK> key. Second, if any command terminates in error, for example if you attempted to display the previous sector when already positioned at the first sector of the disk, an asterisk will be displayed after the **Command:** prompt to indicate that an error has occurred.

### List of DED Commands

<A>	Enter the ASCII modify mode.
<C>	Search for an ASCII string.
<D>	Select a new drive.
<F>	Search for a hexadecimal string.
<G>	Go to next occurrence of search string.
<H>	Enter the hexadecimal modify mode.
<L>	List the current sector to the printer.
<N>	Position to next cylinder.
<O>	Output a TOF (ASCII X'0C') to the printer.
<P>	Position to the previous cylinder.
<R>	Reposition to cylinder and sector.
<S>	Save the sector buffer to the disk.
<V>	Verify sectors.
<X>	Exit to DOS Ready.
<;>	Position to next sector.
<->	Position to previous sector.
<ENTER>	Display menu of commands.
<BREAK>	Cancel current command.

### Cursor Movement

Left Arrow	Move the cursor to the left one position.
Right Arrow	Move the cursor to the right one position.
Down Arrow	Move the cursor down one line.
Up Arrow	Move the cursor up one line.
Shift Left Arrow	Move the cursor to the start of the current line.
Shift Right Arrow	Move the cursor to the end of the current line.
Shift Up Arrow	Move the cursor to the start of the sector buffer. (Relative byte X'00')
Shift Down Arrow	Move the cursor to the end of the sector buffer. (Relative byte X'FF')
@nn	Depress the @ key followed by the two hex digits representing the relative sector position to move the cursor.

## PRO-ESP - Enhanced System Package

### Movement Commands

#### <;> Next sector

Depress the <;> key to advance to the next sector on the disk. If the last sector of the current cylinder is displayed DED will advance to the first sector of the next cylinder. If the last sector of the disk is being displayed DED will place an \* in the command field to indicate that it cannot advance to the next cylinder since it does not exist.

#### <-> Previous sector

Depress the <-> key to reposition to the previous sector on the disk. If the first sector of the current cylinder is being displayed DED will position to the last sector of the previous cylinder. If the first sector of the disk is being displayed DED will place an \* in the command field and will ignore the command.

#### <N> Next cylinder

Depress the <N> key to advance to the next cylinder on the disk. If the last cylinder of the disk is being displayed an \* will be displayed in the command field and the command will be ignored.

#### <P> Previous cylinder

Depress the <P> key to reposition to the previous cylinder on the disk. If the first cylinder on the disk is being displayed an \* will be displayed in the command field and the command will be ignored.

#### <R> Reposition

Depress the <R> key to reposition to a desired cylinder and sector. DED will prompt for the cylinder and then for the sector; enter them as decimal quantities. Depress <Break> to abort and remain positioned at the current cylinder and sector. If a cylinder or sector is entered which does not appear on the disk, you will be reprompted for the cylinder or sector.

### Modification Commands

#### <H> Hexadecimal modify

Depress the <H> key to enter the hexadecimal edit mode. To make changes to the sector buffer, position the cursor over the byte to be changed using the arrow keys. Any digits entered will be interpreted as hexadecimal values. Characters which are not legal hexadecimal digits will be ignored. The hex modify mode can be exited at any time by depressing the <BREAK> key. Remember that any changes made to the sector buffer will not be written to the disk until you issue a <S>ave command.



## PRO-ESP - Enhanced System Package

While in the hexadecimal modify mode, an additional command is available. Depress the <Z> key and DED will zero the remainder of the sector buffer following the byte at the cursor position.

### <A> ASCII modify

Depress the <A> key to enter the ASCII modify mode. To make changes to the sector buffer, position the cursor over the byte to be changed using the arrow keys, and type the new text. As the text is entered the cursor will automatically advance to the next position in the sector buffer. The ASCII modify mode can be exited at any time by depressing the <BREAK> key. Remember that any changes made to the sector buffer will not be written to the disk until you issue a <S>ave command.

While in the ASCII modify mode, the @ cursor positioning command will not function.

### <S> Save sector

Depress the <S> key followed by <ENTER> to save the current sector to the disk. Any modifications made in the sector buffer will not be written to the disk until you <S>ave them.

## Search Commands

### <C> Find ASCII string

Depress the <C> key and DED will prompt for an ASCII string. The ASCII search works exactly like the hexadecimal search described below.

### <F> Find hexadecimal string

Depress the <F> key to search for a hex string. You will be prompted to enter the hex string. After the string of hexadecimal digit pairs has been entered DED will begin searching the disk for a match. The search will start at the byte following the current location of the cursor. If your cursor is positioned at relative byte X'40' on drive 1 cylinder X'00' sector X'00', the search will start from drive 1, cylinder X'00', sector X'00', relative byte X'41'. The search may be aborted at any time by depressing the <BREAK> key. If a match is found, DED will display the sector where the match is found with the cursor positioned to the first byte of the matching string. String search is not affected by sector boundaries. If you wish to search for another occurrence of the same string use the G command.

### <G> Goto next occurrence

Depress the <G> key to go to the next occurrence of the search string.

## PRO-ESP - Enhanced System Package

### Miscellaneous Commands

#### <D> Select drive

Depressing the <D> key followed by the <ENTER> key will cause DED to prompt for a drive number. Enter the drive number you wish to scan or <BREAK> to return to DOS Ready.

#### <X> Exit to DOS

Depress the <X> key followed by the <ENTER> key to return to DOS Ready.

#### <V> Verify sectors

Depress the <V> key and DED will prompt for a sector count. Enter a decimal number less than or equal to 999. DED will then attempt to read that many sectors to verify that they are indeed readable. A running count of the unreadable sectors will be displayed. The verify command may be aborted at any time by depressing the <BREAK> key. After the verification has been completed DED will redisplay the original cylinder and sector.

#### <O> Output top-of-form

Depress the <O> key and DED will send a top-of-form (ASCII X'0C') to the printer.

#### <L> List sector

Depress the <L> key followed by the <ENTER> key and DED will list the contents of the current sector to the printer in the following format:

Drive 01	Cylinder 00	Sector 00
0123456789ABCDEF Byte	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	
..&!.....&\$..	<00> 00 1E 26 21 00 10 01 0F 03 CD BB 04 26 24 01 01	
....&.....!p.:	<10> 01 CD BB 04 26 14 04 0C CD BB 04 FD 21 70 14 3A	
...o.Ad.I. ....E	<20> D8 07 CB 6F 01 41 64 11 49 1D 20 05 CB A8 11 45	
..w ....o.' ..\$	<30> 11 CB 77 20 0C CB 80 CB 6F 11 27 0F 20 03 11 24	
.....w..q..r.	<40> 09 FD 7E 03 E6 03 B0 FD 77 03 FD 71 04 FD 72 07	
.s.:...w. _.&#..	<50> FD 73 08 3A 02 04 FD 77 09 5F 06 00 26 23 CD BF	
..K.#x. ....w.>"	<60> 04 ED 4B CC 23 78 E6 20 FD B6 04 FD 77 04 3E 22	
..w.%.....:.#..2	<70> 81 FD 77 06 25 06 04 CD BF 04 3A 00 23 E6 10 32	
.\$!.....>..2...2.	<80> 0F 24 21 00 13 06 10 3E 10 90 32 E0 07 7E 32 E1	
.#...>.!..U].>...	<90> 07 23 10 F3 3E 04 21 00 15 55 5D F5 3E 09 F1 01	
.....>.....>....	<A0> 00 04 CD D6 04 3E 06 CD D6 04 0B 3E 02 CD CE 04	
...>.....\$. ....H	<B0> 11 00 0C 3E 01 CD CE 04 C3 00 24 1E 00 18 01 48	
.W...!.....	<C0> AF 57 FD E5 FD 21 D8 07 CD 03 00 FD E1 C9 D5 E1	
.2..6 ..2.....	<D0> 13 32 DC 07 36 20 D5 C5 32 DC 07 ED B0 C1 D1 C9	
.....	<E0> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
.....	<F0> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

## PRO-ESP - Enhanced System Package

### DOEDIT/FLT

The DOEDIT filter will allow the editing of video information and will allow this edited information to be passed back through the \*KI device.

Set \*devspec [to] DOEDIT/FLT (parm, parm...)  
Filter \*KI [using] \*devspec

NOCR=ddd            Character to inform DOEDIT not to  
                     send a Carriage Return after the  
                     edited text.

CURSOR=ddd          Character to use as the cursor when  
                     DOEDIT is active.

ACTIVE=ddd          Keystroke to use as the edit  
                     activation character.

abbr: ACTIVE=A, CURSOR=C, NOCR=N [Use DE for devspec]

Note that all parameters may be entered in decimal (ddd), hexadecimal (X'nn'), or ASCII ("a") format.

#### Parameter Functions

##### NOCR=value

If this character is one position to the left of the cursor when <ENTER> is depressed, NO carriage return (X'0D') will be sent through the \*KI device. This parameter defaults to 127 decimal (X'7F').

##### CURSOR=value

This character will be displayed as the cursor while in edit mode. It will not change your normal system cursor! This parameter defaults to 95 decimal (X'5F').

##### ACTIVE=value

Typing this keystroke will activate DOEDIT. This parameter defaults to <CLEAR-SHIFT><E>.

#### DOEDIT Keystroke Functions

<Up Arrow>, <Down Arrow>, <Left Arrow>, and <Right Arrow>

While DOEDIT is active these keystrokes will move the cursor one position up,



## PRO-ESP - Enhanced System Package

down, left, or right, respectively.

<SHIFT><Left Arrow>

This key will move the cursor to the beginning of the current line.

<SHIFT><Right Arrow>

This key will move the cursor to the first space following the last non-blank character in the current line.

<BREAK>

This key will return control to the normal keyboard driver and passes NO characters through the \*KI device. The cursor is restored to its position prior to activation of DOEDIT.

<CLEAR><Right Arrow>

This key will insert one space at the cursor location. All characters on the line to the right of the cursor shift one position to the right.

<CLEAR><Left Arrow>

This key will delete the character under the cursor. All characters on the line to the right of the cursor shift one position to the left.

<CLEAR><Space>

This key will redisplay the previously entered DOS command at the cursor position, without executing it, thus allowing editing of the command line.

<ENTER>

This key will act differently depending upon the presence or absence of the NOCR character, thus:

1) If the character immediately to the left of the cursor when <ENTER> is depressed is NOT the same as the NOCR character, then all characters to the left of the cursor on the current line will be sent through the \*KI device, terminated with a carriage return (X'0D').

2) If the character immediately to the left of the cursor IS the NOCR character, then all characters to the left of the cursor on the current line (excluding the NOCR character itself) will be sent through the \*KI device. The characters will NOT be terminated with a carriage return.

All other keystrokes in the range 32 through 127 (X'20'-X'7F') will overwrite the character under the cursor, and the cursor will move one position to the right.

## PRO-ESP - Enhanced System Package

### FKEY/CMD

FKEY allows you to redefine the codes returned by the function keys.

FKEY (parm, parm, ...)	
F1=nn	Redefine the F1 key.
SF1=nn	Redefine the shifted F1 key.
F2=nn	Redefine the F2 key.
SF2=nn	Redefine the shifted F2 key.
F3=nn	Redefine the F3 key.
SF3=nn	Redefine the shifted F3 key.
DEFAULT	Return keys to DOS defaults.
abbr: DEFAULT=D	

FKEY will allow you to easily redefine your function keys. Specifying the DEFAULT parameter will instruct FKEY to initialize the function keys to their default values of:

```
F1 = X'81' / X'91'
F2 = X'82' / X'92'
F3 = X'83' / X'93'
```

The first number is the unshifted value, and the second number is the value returned if the shift key is depressed along with the function key.

To give you an idea of how FKEY may be used, try the following:

- 1) Install DOEDIT using the default value for the ACTIVE parameter.
- 2) Type FKEY (F1=X'E5')
- 3) Depress the F1 key and see that DOEDIT is now active!

NOTE: The changes made by FKEY will only remain until you reboot; they cannot be SYSGENed.

## PRO-ESP - Enhanced System Package

### IOMON/CMD

IOMON/CMD is a disk drive "filter" which, when installed, will monitor disk input/output for errors and allow you to take corrective action when such occur.

#### IOMON (parm, parm)

ON	Installs the I/O monitor in high memory and enables its operation.
OFF	Disables the I/O monitor's operation and removes it from high memory if possible.
ENABLE	Enables the I/O monitor if it has been previously disabled.
DISABLE	Disables the I/O monitor's operation without removing it from high memory.
TABLE	Displays the currently enabled disk drivers in a tabular format.

Abbr: ON=YES or Y, OFF=NO or N, ENABLE=EN or E,  
DISABLE=DIS or D, TABLE=TAB or T.  
Defaults: ON and TABLE.

The I/O monitor is installed with the simple command:

#### IOMON (ON)

IOMON will "filter" all currently enabled disk drives with its own error trapping routine. Any disabled disk drives will remain unfiltered so that they may be enabled at a later time, if needed, without the interference of the monitor. The monitor can be temporarily disabled, without removing it from high memory, by the command:

#### IOMON (DISABLE)



## PRO-ESP - Enhanced System Package

At this time, any new drivers or drive "filters" may be installed into the system. To reenable the monitor, execute the command:

### IOMON (ENABLE)

The command:

### IOMON (OFF)

will permanently remove the monitor from the system; its high memory allocation will be released if possible.

Remember, at most one of the four major monitor parameters (ON, OFF, ENABLE, or DISABLE) may be specified in the same command line; any combination of two or more is an error and will cause the monitor installation program to abort. The TABLE parameter may be entered along with any one of the other four.

The TABLE parameter may be used to obtain a display of all currently active disk I/O drivers without affecting the current status of the monitor. Any use of the parameters OFF, ON, ENABLE, and DISABLE will also display the driver table. A typical table follows:

Drive 0 =>	IOMON, X'F3B9'	=>	\$ADH, X'F57E'
Drive 1 =>	IOMON, X'F3B9'	=>	\$FD, X'10A1'
Drive 2 =>	IOMON, X'F3B9'	=>	\$FD, X'10A1'
Drive 3 =>	IOMON, X'F3B9'	=>	\$FD, X'10A1'
Drive 4 =>	IOMON, X'F3B9'	=>	\$FD, X'10A1'
Drive 5 =>	Inactive		
Drive 6 =>	Inactive		
Drive 7 =>	Inactive		

This table, for each drive, indicates whether the monitor is active and where in high memory the monitor resides, plus the next driver in the chain and where it resides. The system floppy disk driver is indicated by the module name "\$FD". Any inactive drive is shown as such.

When a trapped disk I/O error occurs, and the monitor is active, the following will be displayed:

```
Disk I/O error X'nn': <message>
Function X'nn', Drive n, Cylinder X'nn', Sector X'nn', Buffer X'nnnn'
<R>etry, <C>ontinue, <I>gnore, <A>ort?
```

As shown, the error code, driver function, drive number, cylinder number, sector number, and buffer address will be displayed, along with a short explanatory message. A list of the driver function codes is available for reference in one of two publications: "The Programmers Guide to LDOS/TRSDOS Version 6" from MISOSYS, page 3-46; or the "TRS-80 Model 4 Technical Reference Manual", from Radio Shack, page 195. The following errors are trapped:

## PRO-ESP - Enhanced System Package

Code	Short Message	Explanation
X'01'	Read Hdr - CRC	Parity error during header read
X'02'	Read - Seek	Seek error during read
X'03'	Read - LD	Lost data during read
X'04'	Read - CRC	Parity error during read
X'05'	Read - RNF	Data record not found during read
X'09'	Write Hdr - CRC	Parity error during header write
X'0A'	Write - Seek	Seek error during write
X'0B'	Write - LD	Lost data during write
X'0C'	Write - CRC	Parity error during write
X'0D'	Write - RNF	Data record not found during write
X'0E'	Write - Flt	Write fault on disk drive
X'0F'	Write - WP	Write protected disk

At this point you must choose one of the four options listed in the third line of the error display. Simply type the first letter of the desired action to be taken, either "R", "C", "I", or "A". Lowercase is accepted.

The <R>etry option will issue the I/O command to the disk driver again in the hope that a retry will be successful. Note that the driver has already attempted the I/O operation several times without success (the exact number of automatic retries is defined by the system variable RFLAG\$). Manual retries with the <R> option are most effective for recovery from parity, lost data, and record not found errors, especially when you are executing a BACKUP or COPY operation.

The <C>ontinue option will exit the I/O monitor, passing the error code unchanged back to the calling program. Many system programs, such as the FORMAT utility, normally expect to see certain kinds of disk I/O errors, and contain routines which will handle the errors automatically. Use the <C> option in these cases.

The <I>gnore option will exit the I/O monitor and return to the calling program as if no error ever occurred. Note that use of this option will prevent the calling program from detecting the error; and that the I/O buffer will probably contain invalid data. The <I> option should only be used when repeated <R>etries have been unsuccessful, and you wish the executing program to continue operation without aborting (for instance, you may wish to continue a BACKUP by class when an error occurs while copying one of the files being backed up).

Finally, the <A>ort option will immediately return to DOS Ready through the system abort routine. This will cancel any executing JCL.

## PRO-ESP - Enhanced System Package

### MINIDOS/FLT

The MINIDOS filter allows you to access some DOS commands without the necessity of being at the DOS Ready level.

Set \*devspec [to] MINIDOS/FLT  
Filter \*KI [using] \*devspec

There are no parameters. [You may use MD for devspec]

After the MINIDOS filter has been installed, simultaneously depress the <CLEAR> key, the <SHIFT> key and one of the following action keys: <C>, <D>, <F>, <K>, <P>, <Q>, <R>, or <T>.

<C> - The C command will toggle the state of the clock display. This is the same as using the TIME (CLOCK=ON) or TIME (CLOCK=OFF) library commands.

<D> - The D command will cause the system to load and execute the system debugger. You may return to the executing program by depressing G<ENTER>.

<F> - The F command will allow you to display the free space remaining on a diskette, in decimal Kilobytes. At the prompt {f}, type in the number of the drive whose free space you wish to view and depress <ENTER>.

<K> - The K command will allow you to remove a file from a disk. At the prompt {k}, enter the name of the file you wish to remove and depress <ENTER>. If an error occurs the appropriate error message will be displayed.

<P> - The P command will allow you to send a byte to the printer device (\*PR). At the prompt {p}, enter the DECIMAL value of the byte you wish to send to the printer, and depress <ENTER>.

<Q> - The Q command will allow you to display the directory of a diskette. At the prompt {q}, enter the number of the drive whose directory you wish to view. You may limit the display by suffixing the drive number with a slash (/) followed by an up to three character file extension. Depress <ENTER> to finish your input.

<R> - The R command will allow you to rename a file. At the first prompt {r}, enter the current name of the file you wish to rename. At the second prompt {R}, enter the new filename.

<T> - The T command will send a Top-of-Form character (ASCII X'0C') to the printer device. If your printer is not capable of executing a hardware formfeed you will have to install the FORMS filter, too.

Note that any MINIDOS function which generates a prompt may be aborted by striking the <BREAK> key.



## PRO-ESP - Enhanced System Package

### NAME/CMD

The NAME utility allows you to change the name and/or date of a diskette.

NAME :d (parm, parm)

:d                      Indicates the drive containing the diskette to be changed.

NAME="new name"      Changes the specified diskette's name to "new name".

DATE="mm/dd/yy"      Changes the specified diskette's date to "mm/dd/yy".

DATE=ON/YES/Y        Changes the specified diskette's date to the current system date.

abbr: NAME=N, DATE=D. There are no defaults.

To change the name of a diskette, use the NAME parameter. Any printable ASCII characters (32-127) are acceptable. The name may be one to eight characters long; it will be padded with spaces to the right to make a field that is eight characters in length. To change the date of a diskette, use the DATE="mm/dd/yy" parameter. Slashes must be used to delimit the month, day, and year fields. Alternatively, the DATE=ON parameter may be used to place the current system date onto the diskette.

An invalid drive, name, or date specification will be flagged appropriately, and NAME will abort. If no parameters are entered, NAME will abort with the message, **Nothing done!**

## PRO-ESP - Enhanced System Package

### PRTOGGLE/CMD

PRTOGGLE allows you to dynamically link the video device to the printer device with a single keystroke.

#### PRTOGGLE/CMD (parm)

ACTIVE=ddd      Set activation keystroke.  
Defaults to <CLEAR-SHIFT><L>.

abbr: ACTIVE=A

The ACTIVE parameter is used to set the activation keystroke for PRTOGGLE. This keystroke may be entered in decimal (ddd), hexadecimal (X'nn'), or ASCII ("a") format.

The PRTOGGLE filter will monitor the keyboard (\*KI) device for depression of the activation keystroke. Upon receipt of the activation keystroke, subsequent information directed to the video (\*DO) device will also be copied to the printer (\*PR) device. This "link" will remain active until the activation keystroke is again depressed.

PRTOGGLE can only be applied at DOS Ready. Also, it creates the phantom devices \*PK and \*PD to implement its filters on the \*KI and \*DO devices, respectively. If either of these phantom devices is already active in the system, PRTOGGLE will not install itself, displaying an error message and returning to DOS Ready. Finally, PRTOGGLE will abort its installation process if no device space is available in the system (i.e. two free device control blocks are required); an appropriate message will be displayed.

## PRO-ESP - Enhanced System Package

### RD40/CMD

RD40/CMD is a disk drive "filter" which, when installed, will allow the reading of a 40-cylinder diskette in an 80-cylinder drive.

#### RD40 :d (parm)

:d	Specifies the number of the 80 cylinder drive which will be used to read the 40 cylinder diskettes.
ON	Installs the RD40 filter in high memory and enables its operation.
OFF	Disables the RD40 filter's operation and removes it from high memory if possible.
ENABLE	Enables the RD40 filter if it has been previously disabled.
DISABLE	Disables the RD40 filter's operation without removing it from high memory.

Abbr: ON=YES or Y, OFF=NO or N, ENABLE=EN or E,  
DISABLE=DIS or D.  
Defaults: ON.

The RD40 filter is installed with the simple command:

RD40 :d (ON)

RD40 will "filter" the 80-cylinder drive specified by ":d", allowing read-only access to 40-cylinder diskettes in that drive. The drive must be enabled in the system. The RD40 filter can be temporarily disabled, without removing it from high memory, by the command:

RD40 :d (DISABLE)

At this time, 80-cylinder diskettes may again be read from and/or written to in the specified drive. To re-enable the RD40 filter for 40-cylinder accesses, invoke the command:

RD40 :d (ENABLE)

The command:

RD40 :d (OFF)



## PRO-ESP - Enhanced System Package

will permanently remove the RD40 filter from the system; its high memory allocation will be released if possible.

Remember, at most one of the four RD40 filter parameters may be specified in the same command line; any combination of two or more is an error and will cause the RD40 filter installation program to abort.

RD40 may be installed independently on more than one drive; each time a new high memory allocation will be made. The module name will be different for each drive; installation on drive 3, for instance, would generate the module name "RD403". Also, every time RD40 is invoked for a particular drive, with any one of the four permissible parameters, the drive will be restored to cylinder 0 in preparation for further disk accesses to that drive. It is recommended that any newly mounted diskette in the RD40-filtered drive be logged in with either the DEVICE or LOG command before any I/O is attempted. Finally, the disk I/O monitor (IOMON/CMD) must be applied after the RD40 filter is established to prevent any spurious errors from being trapped during access of a 40-cylinder diskette.

## PRO-ESP - Enhanced System Package

### UNREMOVE/CMD

The UNREMOVE utility allows you to recover a file which was inadvertently removed from a disk.

**UNREMOVE filename/ext:d**

**filename/ext:d** - The complete specification of the file you want to restore.

By using UNREMOVE you will be able to recover the file, provided that you haven't already reused the disk space in some other file(s).

If more than one removed file is found on the specified disk with the specified name, then all of the matching filenames along with their creation dates will be displayed. Select which file you wish to unremove by depressing the number next to the file with the desired creation date.

If an error occurs during the execution of UNREMOVE, one of the following errors will be displayed:

**Missing drive number!** - A filename is entered which does not contain a drive number

**Can't restore file. Granule(s) ALREADY allocated!** - UNREMOVE determined that some of the disk space for the file has already been reused.

**Can't restore file. Attempt to allocate beyond end of disk!** - UNREMOVE determined that a granule requested in the directory entry does not exist on the disk. This could occur if the directory entry has been damaged. If this is the case, the directory entry must be corrected before UNREMOVE can properly restore the file. Information in the "Programmer's Guide", section 4, should help you in use of the DED disk editor utility if you attempt to fix the problem.

**File NOT found!** - The filename cannot be found on the disk. This could occur if you have specified the wrong disk or if the directory slot has already been reused by another file.

**File ALREADY exists!** - The filename is already on the disk as an active file. If you wish to restore the original file, first rename the active file and use UNREMOVE again.

## PRO-ESP - Enhanced System Package

### XONXOFF/FLT

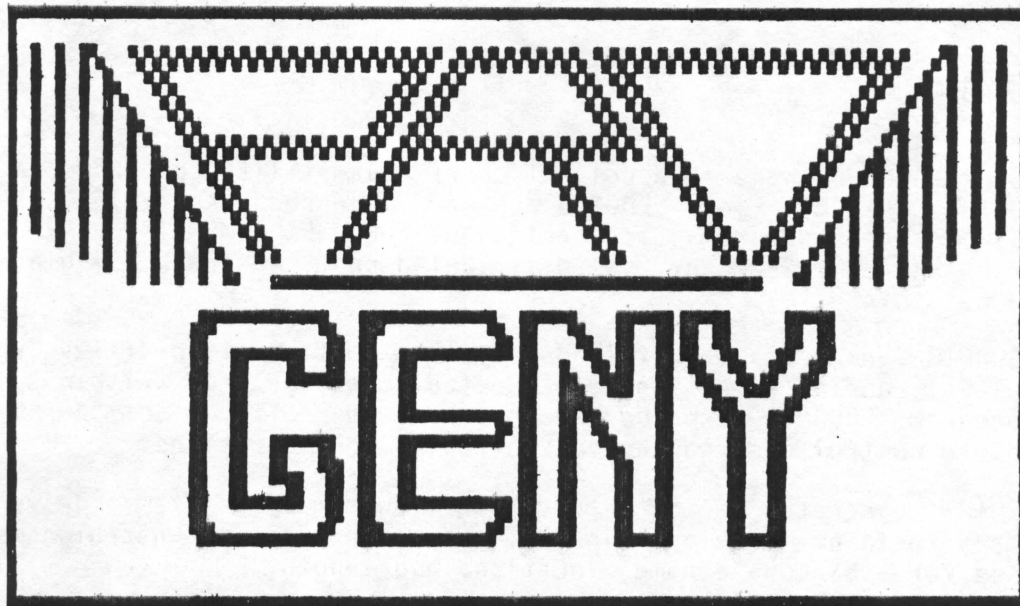
The XONXOFF filter implements the standard XON-XOFF handshaking protocol for those serial devices which require it.

```
Set *devspec1 [to] XONXOFF/FLT
Set *devspec2 [to] COM/DVR
Filter *devspec2 [using] *devspec1
```

There are no parameters.

This filter is designed to allow the attachment of serial output devices (such as printers, terminals, or plotters) which require your host computer to honor the XON-XOFF flow control protocol. These devices will send an XOFF (Control-S or X'13') to the host when they can no longer accept incoming data, and will signal the host to resume transmission with an XON (Control-Q or X'11'). The XONXOFF/FLT can be applied to the communications line device in order to support this handshaking requirement. The filter will operate correctly regardless of baud rate, parity, or word size.





## TABLE OF CONTENTS

=====

GENERAL . . . . .	2
DOCONFIG/CMD * . . . . .	3
MEMDIR/CMD **. . . . .	5
PARMDIR/CMD * . . . . .	7
SWAP/CMD * . . . . .	18

\* authored and Copyrighted (C) 1982 by Roy Soltoff

\*\* authored and Copyrighted (C) 1982 by Scott A. Loomer.

PRO-GENY is published by MISOSYS, PO Box 4848, Alexandria, VA 22303-0848.

LDOS is a trademark of Logical Systems, Inc.

TRSDOS is a trademark of Tandy Corp.

## GENERAL

### GENERAL

=====

The PRO-GENY package is a collection of four utility programs to further enhance the use of your LDOS-6.0 (or licensed equivalents such as TRSDOS-6.0). These programs are entitled: DOCONFIG, MEMDIR, PARMDIR, and SWAP. The PRO-GENY package is distributed on a 40-track single density LDOS-6.0 data diskette.

DOCONFIG generates or reloads system configuration files while at command level, during executing Job Control Language, or within a running BASIC program. DOCONFIG expands the power in the "SYSGEN" command by giving you complete control over saving and restoring configurations.

MEMDIR generates a directory of low-memory system drivers and high-memory resident modules. Finally, you can understand what high memory is being used for - by module name, location, and length.

PARMDIR is essentially a Job Control Language generator or report writer that uses the on-line disk directories as a data base of information. PARMDIR's flexibility will assist you in creating extensive JCL files without having to type in reams of data. You can even construct customized directory listings.

Finally, SWAP provides the facility of reassigning logical-to-physical drive assignments already existing in the system's Drive Code Table.

MISOSYS will continue to supply PRO-fessional "Serious Software (tm)" packages to enhance the operation and convenience of your LDOS-6.0 product. Insist on excellence in software! Get it with MISOSYS Serious Software.

## DO CONFIGURATIONS

### DOCONFIG

=====

This program expands the power of the DOS "SYSGEN" command by giving you much greater control and flexibility in creating or restoring system configurations. The syntax is:

```
=====
|
| DOCONFIG filespec/CFG (SYSGEN)
|
| filespec - is the file to save or restore.
|           The file extension will default
|           to "CFG" if omitted
|
| SYSGEN   - is specified to save a system
|           configuration. If omitted, the
|           system will be restored to the
|           configuration of "filespec"
|
| abbr: SYSGEN=S
|
=====
```

DOCONFIG is a major enhancement of the configuration capabilities of your DOS. DOCONFIG works in one of two ways. You can SAVE the current configuration of your system to ANY file of your choice on any drive of your choice. You can also restore the machine's configuration at any time from any of the configuration files you created. The configuration file is constructed similar to the DOS CONFIG/SYS file (with minor exceptions), except that now YOU control configurations without having to re-boot your machine.

DOCONFIG can function from "DOS Ready" or from @CMNDR execution. It can also be executed from a Job Control Language file to either SAVE or RELOAD a configuration file while the JCL is executing. This will work even if a re-loaded configuration changes the drive assignment for the drive currently executing the JCL file - be it the system's SYSTEM/JCL file or your own execute-only JCL file. DOCONFIG is smart enough to correct the JCL interfacing being done by DOS if drive assignments are switched. If the JCL is SAVING a configuration, the CONFIG file will not reflect JCL as being active. This means that the resulting configuration can be reloaded without reentering the JCL that was currently executing. The use of DOCONFIG now gives JCL more power to run job streams that require revised high-memory configurations for selected applications. Wow, dynamic reconfiguration - on the fly!

You can even execute the DOCONFIG program for saving a configuration while running a BASIC program. This is achieved by executing the command as:

SYSTEM"RUN DOCONFIG filespec (S)"



## DO CONFIGURATIONS

Saving the state of the executing BASIC program will require a 52K configuration file. Restoring the saved configuration file by DOCONFIG at some future time will result in continuing the execution of the BASIC program at the statement following the <SYSTEM"RUN DOCONFIG ...>. The reloading of the configuration is normally done at DOS command level. This can also be done from within BASIC via another <SYSTEM"RUN DOCONFIG> command; however, your current BASIC state will be overwritten unless previously saved with a "DOCONFIG".

Please note that when saving a configuration, the standard DOS message:

### User configuration built

will be displayed at the conclusion of saving the file. This is perfectly normal as DOCONFIG interfaces with and actually executes the DOS library module used to create a CONFIG/SYS configuration file.

A good reason to employ the power of DOCONFIG is to give you an easy means of swapping high-memory configurations - without having to re-boot your machine. Once you establish a particular configuration, say with COM/DVR, FORMS/FLT, KSM/FLT, ..., save it into a configuration file using DOCONFIG. You can easily restore your machine to that configuration with another DOCONFIG command. Flexibility is power!

## MEMORY DIRECTORY

### MEMDIR =====

The program, MEMDIR/CMD, provides a directory of system low memory and high memory usage. MEMDIR can be executed from "DOS Ready", from JCL, or from @CMNDR. From "DOS Ready", its syntax is:

```
=====
|
| MEMDIR (PRINT)
|
| Print - send display to printer
|
| abbr: PRINT=P
|
=====
```

Ever wonder what in the world was up in high memory when you execute a MEMORY command and it says HIGH\$=X'E123'? Where did all that memory go? No need to wonder any more. MEMDIR is here to give you a directory of high memory. It tells you what program/module is there, where it resides, and how long it is. MEMDIR also shows you what is located in the system low-memory driver region. MEMDIR gives the information on your high/low memory usage in the following formatted display:

High Memory Directory	HIGH\$ = X'aaaa'	Length = bbbb	
Module	Start Address	End Address	Length
name	X'cccc'	X'dddd'	eeee

aaaa = the address of the top of unprotected memory  
(or beginning of system low memory);  
bbbb = the decimal length of protected memory in bytes  
(or the quantity of bytes used in system low memory);  
name = the name of the module in protected memory;  
cccc = the address of the first byte of the module;  
dddd = the address of the last byte of the module;  
eeee = the decimal length of the module in bytes.

MEMDIR makes use of the front end linkage header protocol as documented in the Technical Reference Manual for 6.0. For MEMDIR to work properly, all modules occupying protected memory must adhere to the system standard header.

If no memory is protected, MEMDIR will advise you of that fact. If a non-standard module is encountered in protected memory, MEMDIR will attempt to locate the next properly headered module. The region occupied by the improperly headered module will be identified as "unknown".

MEMDIR will pause if it fills the screen with the directory; the display will subsequently pause after each screen page. Pressing any key will display the next page of the directory. The PRINT parameter will direct output to the printer in addition to the screen display.

## MEMORY DIRECTORY

An actual example of such a memory directory listing could be similar to the following directory listing:

MEMDIR 6.0.0 - Copyright 1982 Scott A. Loomer. Licensed to MISOSYS

Low Memory Directory		Start = X'08EB'	Length = 2409
Program	Start Address	End Address	Length
\$KI	X'08EB'	X'0BF9'	783
\$DO	X'0BFA'	X'0E4F'	598
\$PR	X'0E50'	X'0E8C'	61
\$FD	X'0E8D'	X'103B'	431
\$HD0	X'103C'	X'1167'	300
\$CL	X'1168'	X'1253'	236

High Memory Directory		HIGH\$ = X'FE19'	Length = 486
Program	Start Address	End Address	Length
\$KSM	X'FE1A'	X'FF11'	248
\$FF	X'FF12'	X'FFFF'	238



## PARAMETERIZED DIRECTORY UTILITY

### P A R M D I R

=====

This utility command is used to generate output based on conditional tests of directory information. The format of the output is completely under user control. PARMDIR can also directly generate a MAP file of data filespecs for use with PRO-PaDS, the Partitioned Data Set utility. The command syntax is:

=====

PARMDIR partspec outputspec (parm,parm,...)

**partspec** is the partial file specification representing the class of files that you want to examine.

**outputspec** is the file or device that is to receive the output. If no file extension is given, the default is /JCL. If the outputspec is omitted, it is prompted for.

A, B, C => are the prefix positional parameters

X, Y, Z => are the postfix positional parameters

INV, SYS, MOD, DATE have the same selection meanings as in the DIR command.

**SORT** The selected directory records will be in sorted alphabetic order, unless this parameter is turned off. Default is ON.

**ENTER** Allows you to change the default logical carriage return from the semicolon to another character. Default is ";" (semicolon).

**FSPEC** Each selected filespec will be written to the output filespec unless this parameter is suppressed (e.g. FSPEC=NO). Default is ON.

**IF** This parameter allows additional tests to be made before a directory record is passed on to be processed.

Parameters continued on the next page.

=====

## PARAMETERIZED DIRECTORY UTILITY

=====

(Parameters continued)

LABEL	This allows you to execute just a portion of a parameter library. If this parameter is used, the first record of the parmlib MUST be a label (ie., "@PARM1", etc.)
MAP	indicates that a PDS MAP data file is to be generated. Default is OFF.
NOTES	Some JCL comment lines are written to the output device specification. One of them is your query, shown in JCL comment format. To suppress JCL comment lines, set NOTES=N. Default is ON.
PARMS	This parameter points to the parmlib in which the rest of the query is located. The parameters in the file can be extended from the PARMDIR command line, but since the parmlib parameters are the last ones read, they can not be over-ridden. If only PARMS is entered without a filespec, you will be prompted for the file specification.
VIDEO	Show the results of your PARMDIR query on the video screen, in addition to the output device spec. Default is OFF.

abbr: DATE=D, ENTER=E, FSPEC=F, INV=I, LABEL=L, MAP=M,  
NOTES=N, SORT=O, PARMS=P, SYS=S, VIDEO=V

NOTE: all parameters and keywords may be in upper or lower case.

=====

### INTRODUCTION

-----

The PARMDIR utility will allow you to access the information in your disk directories and produce formatted reports, files, and even Job Control Language (JCL) files.

### HOW TO GET STARTED

-----

It is simplest to get started in learning the power of PARMDIR by just typing in:

PARMDIR :Ø \*DO

## PARAMETERIZED DIRECTORY UTILITY

You will note that the names of all of the visible files that you have on your :Ø drive were sorted and written to the screen.

The first entry on the command line, after the PARMDIR command name, was entered as ":Ø". This partial filespec tells the program which directory records to look at. This is similar to the way that the DIR command works. You might have entered "/CMD:Ø" as the first parameter, and then the program would have shown you only the visible files with an extension of "/CMD" on the :Ø drive. Another way might have been to use the "NOT" operator in the partial filespec, as in "-/DVR:Ø", to see all files except the drivers that were on your :Ø disk. PARMDIR has no restrictions on the way that you construct the partial filespec. If the drive specification is omitted from the partial file specification, all drives will be searched. You may also use the dollar-sign character (\$) to signify any character. The dollar-sign is sometimes called a wild-card character, since it can represent any character while performing a comparison. This can be used to search for similarly named files, selecting those filenames that agree with the partial filespec except for the positions where the wild-card character says that we don't care what character is in those positions. An example of this would be if you had a series of accounts-receivable files: ARØFØ1Ø, AR1FØ1Ø, AR2FØ2Ø, AR2FØ1Ø, etc. If you used the partial filespec "AR\$F" you would see a listing of all of those files that are currently on-line (notice that the drive number was omitted from the partial file specification).

The second parameter represented where the output was directed to go. This can be a device specification like \*PR or \*DO (or even the \*CL comm-line driver!), or it can be directed to a file. If a file is used without an extension, the default of /JCL will be automatically added to your filespec.

The real power of this utility lies in the third item following the utility name. This parameter string gives you the flexibility to produce a file that meets almost any of your needs.

### OPTIONAL PARAMETERS

-----

There are many optional parameters that have been developed to give you the ability to produce specialized directory listings and files. Some that you should already be familiar with from the DIR command, like INV to allow gathering directory records from files that have been marked invisible. Similarly, SYS will make the systems files available. Please be sure to note that files that have been marked SYS may include files that do not have the file extension of /SYS. The MOD parameter will only let the program look at files that have been modified since the disk's last backup. The output will normally be sorted in filename order, unless you enter "SORT=N" in the parameter string.



## PARAMETERIZED DIRECTORY UTILITY

The DATE parameter has been designed to let you specify a date range or specific date that you wish to examine. To examine the directory records that had been updated on the 3rd of July, 1983, the DATE parameter would be coded as:

```
...,DATE="07/03/83",...
```

The range of dates from the 3rd to the 15th of July would be written as:

```
...,DATE="07/03/83-07/15/83",...
```

Notice that the dates are written with the opening date of the range given first and the last date following the hyphen (-). Both dates are not necessary. If the hyphen is present following a single date, PARMDIR assumes that all directory records written on or after that date are to be examined. If the hyphen precedes the date, all directory records written on or before that date are used. For example,

```
...,DATE="07/03/83-",...
```

would produce output for records dated 3 July, 1983 or later, while:

```
...,DATE="-07/03/83",...
```

would show only those records written on, or before, the 3rd of July, 1983.

The letters A, B and C are prefix parameters, appearing in the output before the file specification, while the letters X, Y and Z are postfix parameters, appearing after the filespec. These prefix and postfix parameters can be used to generate substitution tokens, used in JCL processing. If you just enter "A" in the parameter string, for example, the output will show a #A# before the file specification. An X would generate an #X# token following the file spec. Let's see an example.

```
PARMDIR /CMD:0 DOFILE:1 (A,X)
```

will produce a JCL file that looks something like this.

```
. PARMDIR: /CMD:0 DOFILE:1 (A,X)
#A# DOCONFIG/CMD:0 #X#
#A# MEMDIR/CMD:0 #X#
#A# PARMDIR/CMD:0 #X#
#A# PDS/CMD:0 #X#
#A# U/CMD:0 #X#
//exit
```

This DO file could be used by executing

## PARAMETERIZED DIRECTORY UTILITY

```
DO DOFILE/JCL:1 (A=LIST,X="(HEX)")
```

which would produce the SYSTEM/JCL file:

```
. PARMDIR: /CMD:Ø DOFILE:1 (A,X)
LIST DOCONFIG/CMD:Ø (HEX)
LIST MEMDIR/CMD:Ø (HEX)
LIST PARMDIR/CMD:Ø (HEX)
LIST PDS/CMD:Ø (HEX)
LIST U/CMD:Ø (HEX)
//exit
```

Note the use of the double quotes surrounding '(HEX)' so that the left parenthesis is properly parsed. If the same JCL file had had the "X" token substituted with the "(P)" parameter to the LIST command, after the DO compile had been completed, the SYSTEM/JCL file would have looked like this.

```
. PARMDIR: /CMD:Ø DOFILE:1 (A,X)
LIST DOCONFIG/CMD:Ø (P)
LIST MEMDIR/CMD:Ø (P)
LIST PARMDIR/CMD:Ø (P)
LIST PDS/CMD:Ø (P)
LIST U/CMD:Ø (P)
//exit
```

These positional parameters can be made to hold strings by typing the pre/postfix parameter followed by an equal sign and the string information contained between double-quotes, eg. A="any string". Let's try an example. Execute the following,

```
PARMDIR :Ø *DO (A="LIST ")
```

You will notice that the output directed to the screen by the \*DO device specification will be a series of lines,

```
. PARMDIR :Ø *DO (A="LIST ")
LIST COM/DVR:Ø
LIST COMM/CMD:Ø
LIST EDAS/CMD:Ø
LIST FORMS/FLT:Ø
LIST KSM/FLT:Ø
LIST PDS/CMD:Ø
//exit
```

The first line is a JCL execution-time comment that echos the PARMDIR query, while the last line signals the end of a JCL stream. Notice that the lines in between append all of the visible file names from drive :Ø to the word "LIST ". If this query had been written to a file, you could have typed in "DO = filename" and the JCL processor would have listed each one of these files.

## PARAMETERIZED DIRECTORY UTILITY

The next optional parameter can be quite useful to those of you who use the PRO-PaDS partitioned data set utility. One of PDS's command, APPEND, can accept a list of files that are to be inserted into a PDS file. Using the MAP option, a MAP file that can be used in the APPEND operation is automatically generated. The MAP parameter will assume that all of the selected records are non-CMD files, since PARMDIR will not read through the files looking for the transfer address information.

Let's try another example. Let us assume that a BASIC Cross-Reference utility named "BREF" is available that operates on programs loaded into BASIC. You may have a need to get the variable cross-reference listing from a series of BASIC programs, so you start writing the PARMDIR query.

```
PARMDIR /BAS:1 FILE (A="LOAD "
```

That would produce a JCL file that would look like:

```
LOAD PROGRAM/BAS:1
LOAD PROG1/BAS:1
LOAD PROG2/BAS:1
```

Now BASIC's LOAD is a funny animal, because it expects a double-quote in front of the program name that it is to load. If we put another quote after the LOAD, it would not work, because PARMDIR is looking at the quotes as its property. This presents us with a problem. Also, if you could place a carriage-return after the filename, you would be able to generate the 'SYSTEM "BREF"'. Well, PARMDIR will allow this, but you must be a little sneaky. Any time that you need a carriage-return in your output, you may enter a semi-colon (;) inside the quote marks.

If you wish to pass a semi-colon through to the output, without having it translated into a carriage-return, you may use the ENTER parameter to change the default logical carriage-return to any other character that you don't need by placing the new logical carriage-return character in quotes after the parameter. The ENTER parm can be entered in one of three forms; ENTER=ddd, ENTER='hh' or ENTER="c", where <ddd> represents a three character decimal character from 0 to 255, 'hh' is a two digit hexadecimal character within apostrophe marks, and "c" is any single character within quotation marks.

The situation sometimes exists where a character must be passed through that has special significance to either PARMDIR or DO. This can be rectified by using (within the quotation marks following a positional parameter) a per-cent mark (%) followed by two hexadecimal numbers. You can thus enter any character that can't be entered directly. Any file that you create using this method MUST be sent through the JCL compilation phase, since that is where the transformation from hex code to actual character takes place. Remembering that a hex 22 is a double-quote ("), let's go back to our problem.

```
PARMDIR /BAS:1 FILE (A="LOAD%22",X="";SYSTEM%22BREF")
```



## PARAMETERIZED DIRECTORY UTILITY

After execution of this command line, PARMDIR will produce a file that looks like this:

```
. PARMDIR /BAS:1 FILE (A="LOAD%22",X=";SYSTEM%22BREF")
LOAD"PROGRAM/BAS:1
SYSTEM"BREF
LOAD"PROG1/BAS:1
SYSTEM"BREF
LOAD"PROG2/BAS:1
SYSTEM"BREF

.
.
.
//exit
```

### KEYWORDS

The real power of PARMDIR is explained in this section. Until this point, you have only been able to attach character strings to the filename. PARMDIR has been made more useful by the ability to place keywords, representing data items from the directory, into the output. Here is a table of the allowable keywords:

	\$DAT	- date that the file was last written, in the format MM/DD/YY.
*	\$EOF	- end of file offset (1 to 256)
*	\$ERN	- ending record number (0 to 65535)
	\$EXT	- file extension
	\$EXX	- file extension extended
*	\$LRL	- logical record length (1 to 256)
	\$NAM	- file name
	\$NAX	- file name extended
*	\$PRO	- protection level (0 to 7, see text)
*	\$REC	- number of records (based on LRL) (0 to 65535)
*	\$DRV	- drive spec (0 to 7)
	\$VID	- 16-byte volume name and date (\$VID is the same as \$VNM concatenated with \$VDT)
	\$VNM	- 8-byte volume (disk or diskette) name
	\$VDT	- 8-byte volume date (MM/DD/YY)
*	Keyword can be used in the "IF" parameter	

## PARAMETERIZED DIRECTORY UTILITY

The two extended fields, \$NAX and \$EXX, are filled out with blanks if their values are shorter than the eight characters allowed for the filename and the three characters for the file extension. This will be useful in our next project. Using a little ingenuity, you can prepare a file that can then be used to make a diskette library listing. If a file specification is used as the output-spec, every time that PARMDIR is executed, the file will be overwritten, which will only allow us to prepare a single diskette directory. However, if we first route a device such as \*PR to a file, and then execute PARMDIR using this device as output, we will be able to keep executing the program (perhaps using the "repeat last DOS command" function) and keep adding new directory records to the end of the file. Try this:

```
ROUTE *PR DIR/FIL:Ø
```

```
PARMDIR :1 *PR (A="$NAX/$EXX $DAT $VNM",FSPEC=N,VIDEO,NOTES=N)
```

Now, put another diskette in drive 1 and repeat the PARMDIR execution for each of your diskettes. RESET \*PR and you should have a file like this.

```
CMDFILE /CMD 1Ø-Dec-81 DATAØ1A
DATACOMM/KSM 27-Dec-81 DATAØ1A
FILESPLT/ 28-Feb-82 DATAØ1A
BUGT1982/VC 11-Jun-82 DATAØ1B
CAT /JCL 22-Apr-82 DATAØ1B
PARMDIR /CMD 21-Apr-82 PARMDIR
PARMDIR /DOC 15-Jun-82 PARMDIR
JL /DVR 1Ø-Dec-81 LDOSSOLE
KSM /FLT 2Ø-Mar-82 LDOSSOLE
MINIDOS /FLT 2Ø-Mar-82 LDOSSOLE
```

This file (somewhat abbreviated) represents the results of four executions of PARMDIR. Notice that the file has the various filenames, extensions, file dates and the name of the diskette in neat columns. With a little BASIC program, use of a SYSTEM "SORT (VAR="NAME",LEN=LENGTH) sorting utility, a listing can be made with all of the files in filename (or date) order. Never again will you have to wonder about the location of some piece of software or a file!

Look again at the optional parameter. Did you notice the parameters "FSPEC=N,NOTES=N,VIDEO"? Since the output was intended as a data file, rather than a JCL file, the "NOTES=N" turns off the generation of the JCL "//EXIT" command and the comments. If the "FSPEC=N" were not used, each record would be followed by the complete filename. This automatic feature was used when the JCL file was generated that LISTED all of the files on a diskette. Finally, the "VIDEO" parameter was turned on, since the output was going to a file and we wished to also look at it on the screen.

## PARAMETERIZED DIRECTORY UTILITY

### PARMLIBS

-----

If you have been trying the examples as we went along, you will have noticed that the command line for the PARMDIR command rapidly approaches the 79 character limit for keyboard entry. In order to get around this command line size limitation, PARMDIR has the ability to get its parameters from a parameter library (parmlib).

If you wish to use the parmlib facility, enter the PARMS parameter in the optional parm list. You may enter the filename of the parmlib in string format (e.g. PARMS="MYPARMS",) or if you wish to be prompted for the filename of the library, just enter "PARMS". PARMDIR assumes "/PRM" as the default extension for parmlibs.

The parmlib can be constructed using the DOS BUILD command, the PRO-CREATE editor, or any available text processor. You must be certain to save your text file using an ASCII option if you are going to use a text processor as PARMDIR requires proper line-terminating carriage returns.

Parameters can be written in a relatively free fashion. Parms can be written one to a line, followed by a carriage return, or several parameters separated by commas can be written on a line. These parameters can be extended by other entries on the command line, but cannot be modified. An example of this is that you could leave "VIDEO" out of your parmlib entry, and then add it later in the command line by entering "PARMS,VIDEO,etc.". However, if you did have VIDEO in the parmlib entry, you can not turn it off on the command line by using a "VIDEO=N" parameter.

Since the DOS is device independent, it is possible to use PARM="\*KI" to enter the parameters from the keyboard. This can be extremely helpful while testing. The way to signal the end of file to PARMDIR is to enter a logical end of file character. This is usually done by holding down the <CONTROL-SHIFT-@> keys simultaneously Consult your DOS manual for details.

Since the smallest unit of disk drive allocation is the gran and most parmlibs are much shorter than that, having a separate parmlib for each set of parameters can be very wasteful of disk space. PARMDIR has gotten around this problem by allowing several sets of parameters to be combined into one parmlib, therefore making much better use of disk space. When you build your parmlib, if you add a label as the first line of each set of parameters, you can have many different sets of parameters contained in one parmlib. A label is formed by the at-sign ("@") followed by up to 15 alphabetic and numeric characters, however a more practical limit should be eight characters following the at-sign. Here is a sample of a parameter library:

```
@wrtlist
a="$NAX $EXX $drv $dat $rec $LRL $eof"
if="$pro > read"
@DIR056
a="$nax $exx $drv $dat $rec $lrl"
if="$drv = 0 + $drv = 5 + $drv = 6",f=n
```



## PARAMETERIZED DIRECTORY UTILITY

You should notice that PARMDIR is not sensitive to the case in which the parameters are written. The "IF" parameter is covered in the next section.

### CONDITIONAL EXECUTION (IF)

Earlier, you learned that it is possible to selectively look at files, depending on the partial filespec. The "IF" parm further allows you to conditionally accept or reject files, based on logical comparisons within the "IF" parm. In the table accompanying the keyword section, each one of the keywords that have been marked with an asterisk can be used in comparisons with data elements from the directories.

The comparison operators are similar to the ones used in BASIC and Job Control Language. They are:

<	Less than
>	Greater than
=	Equal to
<>	Not equal to
<= or =<	Less than or equal to
=> or >=	Greater than or equal to
&	logical AND
+	logical OR
-	logical NOT

Each of the data items that can be used with the "IF" statement, except for \$PRO (protection level), are numeric. \$PRO can be tested for the actual words representing the protection level (CANT (none), EXEC, UPDT, READ, WRIT, NAME (rename), MOVE (remove), or FULL,) or the number (0-7) that stands for the access level as shown below. Abbreviation to first character of the word is acceptable.

0-FULL	1-MOVE	2-NAME	3-WRITE
4-UPDATE	5-READ	6-EXECUTE	7-CANT

The IF="expression" can be compound and spaces between tokens are acceptable. Logical expressions can be connected as in JCL by using the symbols for AND, OR, and NOT (&, +, and -). An example of a compound "IF" expression is,

## PARAMETERIZED DIRECTORY UTILITY

IF=" \$LRL <= 10 & \$REC >= 60000"

This would take all files that met the partial filespec comparison and would additionally check to see that the file had a logical record length (LRL) less than or equal to 10 and also that there were 60,000 or more records in the file, before it would accept the directory record for eventual output.

In the PARMLIB section above, again look at the sample parmlib. It contains two sets of parameters, labelled WRTLIST and DIR056. Referring back to the list of keywords, let's look at these members of the parmlib. Both of them create the output format using the "A" prefix parameter, but the "IF" parameter causes different sets of filenames to be examined and selected from the on-line directories. WRTLIST will accept only those directory records that meet the partial filespec from the command line and also only those files whose protection status is NAME, MOVE or FULL (ie., greater than WRIT.) DIR056 was written to output the formatted directory information only for those files that were contained on drives (\$DRV) 0, 5 and 6. Obviously, DIR056 was written for someone that had more than four drives, but who wasn't interested in the files contained on drives 1, 2, 3, or 4.

### ERROR PROCESSING

-----

There are two error types in PARMDIR. **Syntax error** means that something is wrong with the conditional "IF" expression, while the **Type mismatch** error occurs when a value field is a string and the command parser expects a number. In case of an error, the optional parameters will be listed on one line with the error being highlighted on the next line by a line of dots with a dollar-sign (\$) under the error.

**NOTE:** In the case of a compound "IF" statement using AND logic, the first FALSE term makes the entire statement FALSE, therefore the parser will stop examining the rest of the statement. Thus, it is possible for an error in syntax or type to be un-flagged if it occurs in a term subsequent to an AND-connected FALSE term. The PARMDIR output, however, will still be correct.

## DRIVE REASSIGNMENT

### S W A P

=====

This program expands the power of the DOS "SYSTEM (SYSTEM=d)" command by allowing you to reassign the logical-to-physical drive assignments for any two drives even if Job Control Language is under execution. The syntax is:

```
=====
|
|  SWAP :s :d
|
|  :s      - is the drive specification of
|            the SOURCE drive. It may be in
|            the range [0-7 & <> :d].
|
|  :d      - is the drive specification of
|            the destination drive. It may
|            be in the range [0-7 & <> :s].
|
|=====
```

This utility is used to reassign the relationships between a logical drive number (specifications 0 through 7) and the physical disk drives associated through the system's Drive Code Table (DCT). SWAP will function from "DOS Ready", from JCL, or from @CMNDR execution (i.e. SYSTEM "RUN SWAP :s :d"). SWAP is especially useful within Job Control Language files that are used to install particular system configurations.

If you attempt to swap the SYSTEM drive (drive 0), the replacement drive must contain a system disk. SWAP will prompt for a system disk if necessary and able; however, if SWAP is being executed from JCL, it will abort.

If JCL is executing from one of the drives being swapped, SWAP will reassign the logical drive number used for the JCL to match the reassignment.

An example of a swap command is:

**SWAP :4 :1**

which will exchange the physical drive associated with logical drive 4 with that associated with logical drive 1. Note that what was referenced as drive 4 is now referenced as drive 1 and vice versa.



# MACH2 - File Space Allocation Utility



Copyright 1983 by Karl A. Hessinger - MicroConsultants, All rights reserved  
Published by MISOSYS, Alexandria, Virginia

## TABLE OF CONTENTS

General Information . . . . .	2
Distribution Diskette . . . . .	2
Invoking MAPPER . . . . .	3
Invoking ALLOC . . . . .	5
Invoking CALC . . . . .	7
Invoking HANDY . . . . .	8
Space Optimization Procedures . . . . .	10
Glossary . . . . .	14

Note: LDOS is a trademark of Logical Systems Incorporated  
TRSDOS is a trademark of Tandy Corp.

## MACH2 - File Space Allocation Utility

### GENERAL INFORMATION

=====

The LDOS file system allocates space for your files in one of two ways. It either assigns space randomly [as used in LDOS 5.0.x, 5.1.0, 5.1.1, 5.1.2, 5.1.3, 6.0] or assigns space automatically starting from cylinder one [5.1.4, 6.1]. Since the access of disk files requires a disk drive to step from track to track [which is one of the slowest operations of a disk drive excluding turn-on delay], the most efficient placement of files is one that minimizes this stepping operation. Either method may prove inefficient in particular cases since the optimum arrangement is dependent on the specific function and access sequence of all files on the disk.

Once and for all, MISOSYS now puts you in the driver seat when it comes to DOS allocation of disk space. Whether you are a commercial programmer wanting to construct "optimized" master diskettes for distribution purposes, whether you are a sophisticated hacker wanting to squeeze every bit of performance out of your system, or whether you just want to create the most contiguous files where you want them, MACH2 is for you.

MACH2 is a collection of four utilities that were designed to work together with such ease, that you will be amazed at how easy direct control of space allocation can be. The MACH2 utilities are friendly. The MACH2 utilities are flexible. Finally, the MACH2 utilities are powerful. You are not limited to floppies, MACH2 works just as well with hard drive systems.

We know that you have been asking for better control over space allocation for a long time. We know that you have needed MACH2 for a long time. We know that you do not want more stringent DOS control over file placement. Now you can take control! MISOSYS delivers with MACH2!

### DISTRIBUTION DISKETTE

=====

This documentation covers the operation of both the Model I/III LDOS 5.1 version (MACH2) and the LDOS or TRSDOS 6.x compatible version (PRO-MACH2). The MACH2 package is provided on a 35-track single density data diskette for LDOS Version 5.1. The PRO-MACH2 package is provided on a 40-track single density data diskette for LDOS/TRSDOS Version 6.

## MACH2 - File Space Allocation Utility

### MAPPER

=====

The operating system's directory command (DIR) will let you know what files you have on a disk, but MAPPER will show you where the files are positioned on that disk. It is invoked via the syntax:

```
=====
|
| MAPPER :d (BLANK,PRINT)
|
| :d      - Specifies which drive to map.
|
| BLANK   - Used to generate an allocation worksheet.
|
| PRINT   - Sends output to the printer device.
|
| Abbreviations: B=BLANK, P=PRINT
|
|=====
```

The MAPPER provides a diskette map by granule by file. You can use the MAPPER just to get a look at what files occupy each granule of disk space. This information is great for other uses [such as reconstruction of damaged cylinders]. Use the MAPPER with the BLANK parameter option on a freshly formatted disk [or other such disk with available free space] to obtain a worksheet for manually preparing your optimum arrangement of files.

If you do not enter a drive number on the command line, MAPPER will prompt you to enter the drive number via the prompt:

Drive ? .

The BLANK parameter will cause the free granules on the disk to be displayed as a bracketed blank field, "[ ]" (without the quotes) instead of the normal field entry of "\*\* Empty \*\*". This can be used to generate a worksheet for use in creating an optimized disk by ALLOC.

While the map is being displayed on the video screen, the following keys will allow you to scroll through the map:

Key Code	Key Function
<UP ARROW>	Display previous screen page
<DOWN ARROW>	Display next screen page
<X> or <BREAK>	Exit to DOS

The PRINT parameter option will cause the disk map to be sent to the printer instead of the video display.



## MACH2 - File Space Allocation Utility

MAPPER produces a screen or printer listing similar to the following illustration [some lines have been deleted to abbreviate the listing]:

Diskname : TESTDISK		Free Space : 108K	
Sides : 1		Density : Double	Cylinders : 40
-----			
0	BOOT/SYS	M80/CMD	M80/CMD
1	M80/CMD	M80/CMD	M80/CMD
2	M80/CMD	M80/CMD	TEST/CMD
3	TEST/CMD	TEST/CMD	TEST/CMD
19	** Empty **	** Empty **	** Empty **
20	DIR/SYS	DIR/SYS	DIR/SYS
24	** Empty **	** Empty **	** Empty **
25	** Locked **	** Locked **	** Locked **
26	** Empty **	** Empty **	** Empty **
27	M8OPACK/CMD	M8OPACK/CMD	M8OPACK/CMD
28	M8OPACK/CMD	M8OPACK/CMD	M8OPACK/CMD
37	** Empty **	** Empty **	** Empty **
38	M80/CMD	M80/CMD	M80/CMD

The "file specification field" will contain the string "\*\* error \*\*" if the associated granule is allocated without a file referencing the granule. In addition, if two or more files are assigned to the same granule, that error will be indicated by appending the string "<---" to the file specification.

If you use the BLANK parameter option on a freshly formatted disk with no additional files present, the worksheet generated will look like this [some lines have been deleted to abbreviate the listing]:

Diskname : TESTDISK		Free Space : 174K	
Sides : 1		Density : Double	Cylinders : 40
-----			
0	BOOT/SYS	[	]
1	[	[	]
2	[	[	]
3	[	[	]
4	[	[	]
5	[	[	]
- - - - -			
20	DIR/SYS	DIR/SYS	DIR/SYS
21	[	[	]
22	[	[	]
23	[	[	]
- - - - -			
37	[	[	]
38	[	[	]
39	[	[	]

## MACH2 - File Space Allocation Utility

### ALLOC

=====

Pick your media and use your MAPPER-generated worksheet to develop your customized layout of files - or plunge right in to ALLOC since it gives you a dynamic free-space map on-line! ALLOC will allow you to place a file at any location on a disk that you choose. It is invoked simply by entering:

```
=====
|
|  ALLOC
|
|  There are no parameters.
|
|=====
```

The allocation tool, ALLOC, lets you tell the DOS where you want a file placed in up to four directory extents. ALLOC's screen display conveniently presents the controls. All you need do to start the process is specify the file specification. ALLOC even provides an option to specify the file as being "created", thus inhibiting the DOS from deallocating file space.

You will be prompted for the name of the file you wish to allocate via the prompt:

Filename ? .....

The file specification must include a drive number. Once you identify the file specification, ALLOC gives you a three-line scrolling free-space map so that you can see what granules have already been allocated on the disk in question. The free space map will be displayed to help you plan where the file or files can be located. Look at the useful ALLOC screen display as it prompts for the granule count during a file allocation procedure:

```

      ALLOC - LDOS File Space Allocator

6- 11  xx.    ...    ...    ...    xxx    x..
12- 17  ...    ...    ...    ...    ...    ...
18- 23  ...    ...    xxx    ...    ...    ...

      Allocating file : TEST/DAT:3

      Extent      Starting      Starting      Granule
      =====      Cylinder      Granule      Count
              1              12              0              ..
```

The three-line free space map may be scrolled in-place by using the <UP-ARROW> or <DOWN-ARROW> keys as an entry to any "extent" prompt.

For each of the four possible extents, you will be prompted for the starting cylinder, the starting granule, and the granule count. Each prompt

## MACH2 - File Space Allocation Utility

is identified to you via the position of the cursor. Cylinders and granules are both numbered starting from zero. Thus for example, a three granule per cylinder diskette (5-1/4" double density) has granules numbered 0, 1, and 2. A 40-cylinder diskette has cylinders numbered from 0 through 39. Depressing <BREAK> or <ENTER> to the starting cylinder prompt will instruct ALLOC that you have entered all of the extents. Depressing <BREAK> from either of the two remaining prompts will cause ALLOC to restart the entry of that particular extent.

When information for all of the extents has been entered, you will be provided an opportunity to commence the physical allocation or retreat via the prompt:

OK to allocate/create this file (Y/N/C) ? .

The response to this query also establishes whether or not the file's CREATE bit will be set. If you depress <N>, ALLOC will abort the physical allocation thereby providing you with an "escape" mechanism in case of an entry error. If the allocation is aborted, the filename will be removed from the target disk's directory. Depress <Y>, and ALLOC will physically allocate the file. If you specify <C>, the file will be allocated and the "CREATE" bit will be set in the file's directory entry. The "create" bit is used by the DOS to ensure that the space will not be deallocated. It's primary utility is for the pre-allocation of space for random access files similar to the DOS CREATE library command; however, ALLOC obtains this space using the least number of extents based on your own analysis [if you want this selection performed automatically, use HANDY]. Thus, ALLOC's procedure generates a file that is optimum for access speed.

ALLOC can allocate space for a file that already exists in the directory; however, ALLOC is incapable of allocating space to an existing file which has a non-zero file length [i.e. you can ALLOC space to an existing null length file but not one that already has an allocation of space].



## MACH2 - File Space Allocation Utility

### CALC

====

The CALC utility can process a disk's directory to let you know exactly how many granules each file would take up on most media formats supported by the DOS. This tool is great for taking a disk full of programs that are on one media type and ascertaining the granule requirements for a different media type. CALC is invoked via the syntax:

```
=====
|
|  CALC :d (PRINT)
|
|  :d      - Specifies which drive to calculate.
|
|  PRINT   - Sends output to the printer device.
|
|  Abbreviations: P=PRINT
|
|=====
```

If a drive number is not entered on the command line, you will be prompted to enter the drive number via the prompt:

Drive ? .

CALC will display a listing of all files on the disk along with the number of granules that would be required for that file on various media types. The listing will be normally directed to the video screen but can be redirected to the printer device via the parameter, PRINT.

As easy as CALC :1 (P), you can obtain a listing similar to the following printout:

Filename	5"	5"	8"	8"	==== Hard Disk ====		
	single	double	single	double	4 SPG	16 SPG	32 SPG
=====	=====	=====	=====	=====	=====	=====	=====
BOOT/SYS	1	1	1	1	2	1	1
DIR/SYS	4	3	3	2	5	2	1
M80/CMD	15	13	10	8	19	5	3
M80PACK/CMD	15	12	9	8	18	5	3
TEST/CIM	5	4	3	3	6	2	1
TEST/CMD	15	12	9	8	18	5	3
TEST/DAT	0	0	0	0	0	0	0

The first two columns relate to 5-1/4 inch single and double density media indicating five and six Sectors Per Granule (SPG) respectively. The third and fourth columns reference the granule requirements on eight-inch single and double density media indicating eight and ten SPG respectively. The final three columns are provided for use with hard disk allocation schemes using four, sixteen, or thirty-two SPG respectively.

## MACH2 - File Space Allocation Utility

### HANDY

=====

Want to just get a large block of contiguous space for that data base? HANDY is just handy for those non-critical allocation jobs. HANDY will easily allocate the most contiguous extent of space in up to four extents - the number of extents is controlled by you! HANDY determines where the file will fit on the disk according to your extent specification. It is easily invoked via the command specification:

```
=====
HANDY filespec (SIZE=dd,CREATE)

filespec - is the name of the file that you want to
           allocate space for. The entry is optional.
           If omitted, you will be prompted for the
           name of the file(s).

SIZE=dd - is the amount of space (in K) that you
          want to allocate for the file identified.

CREATE - is specified if you want the directory
         entry for "filespec" of "SIZE=dd" to have
         it's "CREATE" bit set (see text).

Abbreviations: S=SIZE,C=CREATE
=====
```

HANDY operates in two modes. To streamline the allocation of a single file with HANDY, you can enter the file specification and the SIZE parameter directly on the command line. The "CREATE" parameter is used to specify the nature of such a file. If the full command-line entry of information is performed, HANDY will automatically allocate the file space and exit without pausing for any prompts. Alternatively, if the file specification is omitted, HANDY will not accept any parameters from the command line. This method of usage results in going through the menu prompts that will be presented.

After the the program is loaded, if you have not entered the file specification on the command line, you will be prompted with:

Filename ? .....

Be sure that the file specification has a drive number. File space is requested in units of "K" (1024 characters). After the filespec has been entered, you will be prompted to enter the total space to be allocated for the file. This query is:

Size of file (in K) ? ...

HANDY has the ability to restrict the automatic allocation from using specified cylinders. We will use the term "lock". This does not mean that HANDY will permanently restrict such allocation - it will invoke the restriction only for the allocation of the file identified by you. This

## MACH2 - File Space Allocation Utility

feature can be useful if you wish HANDY not to use certain portions of the disk - such as the last 10 cylinders, or the first 5 cylinders. You specify the cylinder ranges to lock in response to the prompts:

Lock cylinders : ...  
through : ...

You may enter as many cylinder ranges as you wish. When you complete an entry, you will be reprompted for an additional range. When you have finished, simply depress <ENTER> at the "Lock cylinders : " prompt.

HANDY calculates the fewest number of extents for the given file size and the target disk, and displays the results. The menu at this point would look something like the following illustration:

```

                                HANDY - LDOS File Space Allocator

                                Filename ? yourfile/dat:3
                                Size of file (in K) ? 12

                                Lock cylinders :
                                through :

Extent          Starting      Starting      Granule
=====          Cylinder      Granule      Count
=====          =====
1                21            0                8

                                OK to allocate/create this file (Y/N/C) ? _

```

When information for all of the extents has been displayed, you will be provided an opportunity to commence the physical allocation or retreat via the prompt:

OK to allocate/create this file (Y/N/C) ? .

The response to this query also establishes whether or not the "CREATE" bit will be set. If you depress <N>, HANDY will abort the physical allocation thereby providing you with an "escape" mechanism. If the allocation is aborted, the filename will be removed from the target disk's directory. Depress <Y>, and HANDY will physically allocate the file. If you specify <C>, the file will be allocated and the "CREATE" bit will be set in the file's directory entry. The "create" bit is used by the DOS to ensure that the space will not be deallocated. It's primary utility is for the pre-allocation of space for random access files; however, HANDY obtains this space using the least number of extents.

HANDY cannot allocate space to an existing file which has a non-zero length or which would require more than four extents.



## MACH2 - File Space Allocation Utility

### Creating an Optimized Disk

=====

Before we discuss a step-by-step method that can be used to construct an optimized disk, it is necessary for you to understand some of the actions that the operating system performs when accessing any given file. First, file access consists generally of up to three parts: the opening of a file, passing data from/to the file (input/output), and closing the file. Depending on the size of the file, it takes up space on one or more cylinders. During the process of opening or closing a file, the DOS must read the disk's directory. The disk's directory is usually kept on a cylinder at the midpoint of the disk (i.e. a 40-cylinder disk has its directory at cylinder 20).

A disk takes a specified period of time to step from track to track. This may vary on 5-1/4" drives, for example, from 6 milliseconds per track (fastest drive) to up to 40 milliseconds per track (slowest drive). Therefore, the distance in tracks of a file from the directory dictates the minimum amount of disk time required before the first file record can be accessed. If the file is 15 tracks away from the directory on a 20ms step rate drive, this adds up to 300 milliseconds.

The space occupied by a file may not exist in a single connected chunk [we actually call this chunk of space an extent]. The size of an extent will vary according to what granules are available when the file is allocated [HANDY will always allocate the space to achieve a maximum number of granules per extent]. An extent cannot exceed 32 granules. Therefore, the number of extents taken up by a file depends on both the size of the file and the "spottedness" of the available free space. When a file is opened, the DOS maintains a file control block (FCB) which contains information on accessing the file's disk records. This FCB can hold the data on up to four extents. Therefore, once a file has more than four extents, the DOS will have to re-examine the file's directory to obtain the data on an extent not found in the FCB. Since the new extent data replaces one of the old, a randomly accessed file with a great number of extents incurs an excessive overhead of directory access which implies greater disk drive head thrashing.

This discussion presents two observations. First, files that are frequently accessed should be placed close to the directory. Second, files that are randomly accessed should be contained in as few extents as possible. With these points in mind, let's illustrate a few steps to be taken in order to construct a diskette for optimum access. In this case, we are going to refer to a "frequently accessed" situation - such as a DOS system disk. The following steps may provide a simple way to create just such an optimized diskette.

1. Select the freshly formatted destination disk and invoke MAPPER with both the BLANK and the PRINT options. This will produce a worksheet that you can use to manually construct the disk layout.
2. Invoke CALC referencing the source disk to calculate the number of granules that each file will require.
3. Make your space selections by writing the names of the files onto the worksheet. Remember that placing the most

### Optimization Techniques

## MACH2 - File Space Allocation Utility

frequently used files closest to the directory should yield the highest performance. This means you have to be aware of the frequency of usage of each file.

4. Using ALLOC, place the files on the disk according to the map you have just manually worked up.

5. Use BACKUP or COPY to transfer the actual files from the SOURCE disk to the allocated DESTINATION disk.

Let's examine a specific example. You want to develop an optimized 5-1/4" 80-track double density system disk with two sides. Your source is a 40-track double density 5-1/4" single sided diskette. For this operation, you will not need CALC. What you want to do is to obtain the existing map of the source (under the assumption that you have been supplied an optimum system disk). Using MAPPER on the source, you will obtain a printout similar to the following [some lines have been deleted to abbreviate the listing]:

Diskname : LDOS-514	Free Space : 62K
Sides : 1	Density : Double      Cylinders : 40
-----	
0 BOOT/SYS	QFB/CMD
1 QFB/CMD	FED/CMD
2 FED/CMD	FED/CMD
3 ** Empty **	** Empty **
7 ** Empty **	** Empty **
8 LOG/CMD	JL/DVR
9 MOD3/DCT	REPAIR/CMD
10 CMDFILE/CMD	KI/DVR
11 FORMAT/CMD	FORMAT/CMD
12 BACKUP/CMD	BACKUP/CMD
13 BACKUP/CMD	LBASIC/OV1
14 LBASIC/OV2	LBASIC/OV3
15 LBASIC/CMD	LBASIC/CMD
16 SYS0/SYS	SYS0/SYS
17 SYS7/SYS	SYS7/SYS
18 SYS7/SYS	SYS7/SYS
19 SYS7/SYS	SYS2/SYS
20 DIR/SYS	DIR/SYS
21 SYS8/SYS	SYS1/SYS
22 SYS11/SYS	SYS12/SYS
23 SYS6/SYS	SYS6/SYS
24 SYS6/SYS	SYS6/SYS
25 SYS6/SYS	SYS6/SYS
26 SYS5/SYS	SYS9/SYS
27 PATCH/CMD	PATCH/CMD
28 MINIDOS/FLT	KSM/FLT
29 LCOMM/CMD	RS232T/DVR
30 EQUATE3/EQU	EQUATE3/EQU
31 COPY23B/BAS	** Empty **
32 ** Empty **	** Empty **
39 ** Empty **	** Empty **

## MACH2 - File Space Allocation Utility

The next step is to invoke MAPPER on the freshly formatted destination disk using the PRINT and BLANK parameters. This will generate a worksheet that looks something like the following [some lines have been deleted to abbreviate the listing while others have been filled in based on the next step that you will undertake]:

Diskname : FLOP		Free Space : 707K		
Sides : 2		Density : Double		
		Cylinders : 80		
0	BOOT/SYS	BOOT/SYS	BOOT/SYS	[ qfb/cmd ]
	[ -----> ]	[ -----> ]		
1	[ fed/cmd ]	[ -----> ]	[ -----> ]	[ -----> ]
	[ -----> ]	[ ]		
2	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
32	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
33	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
34	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
35	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
36	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
37	[ ]	[ ]	[ lbasic/cmd ]	[ -----> ]
	[ -----> ]	[ -----> ]		
38	[ -----> ]	[ -----> ]	[ -----> ]	[ sys7/sys ]
	[ -----> ]	[ -----> ]		
39	[ -----> ]	[ -----> ]	[ -----> ]	[ -----> ]
	[ sys2/sys ]	[ sys3/sys ]		
40	DIR/SYS	DIR/SYS	DIR/SYS	DIR/SYS
	DIR/SYS	DIR/SYS		
41	[ sys8/sys ]	[ sys1/sys ]	[ sys10/sys ]	[ sys11/sys ]
	[ sys12/sys ]	[ sys6/sys ]		
42	[ -----> ]	[ -----> ]	[ -----> ]	[ -----> ]
	[ -----> ]	[ -----> ]		
43	[ -----> ]	[ -----> ]	[ sys4/sys ]	[ sys5/sys ]
	[ sys9/sys ]	[ basic/cmd ]		
44	[ patch/cmd ]	[ -----> ]	[ pr/flt ]	[ ]
	[ ]	[ ]		
45	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
46	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
47	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
78	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		
79	[ ]	[ ]	[ ]	[ ]
	[ ]	[ ]		



## MACH2 - File Space Allocation Utility

The next step is to transfer the file names from the source map to the destination worksheet, working out from either side of the directory. The sample worksheet has already been filled with some of the file names. Since both the source and destination diskettes have the same number of sectors per granule, CALC was not needed to calculate the granule requirements. You would have needed CALC's information if the source and destination used a different number of sectors per granule - say one being a single density 5-1/4" diskette and the other being a double density 5-1/4" diskette.

Once the worksheet is completed, the information is keyed into ALLOC so that the space for each file is allocated. The series of entries will be something like the following:

user entry	description of entry
alloc	invoke ALLOC
sys8/sys:2	enter the file specification
41	starting cylinder = 41
0	starting granule = 0
1	the file uses one granule
<ENTER>	response to second extent
<Y>	OK to allocate
sys1/sys:2	the next file specification
41	starting cylinder = 41
1	starting granule is 1
1	also one granule used
<ENTER>	response to second extent
<Y>	OK to allocate
sys6/sys:2	let's skip to this file
41	starting cylinder is 41
5	starting granule = 5
9	sys6/sys used 9 granules
<ENTER>	
<Y>	
(continue with data for remaining files)	

Model I/III users may prepare a "key" file containing all required entries and use the redirection capabilities of ZSHELL to redirect the keyboard input to use the prepared key file.

Up to this point, all of the files have been allocated space on the destination diskette. You now have to transfer the contents of the existing files from the source diskette to the destination diskette. Use BACKUP to accomplish this procedure. This completes the allocation and generation of the optimized diskette.

## MACH2 - File Space Allocation Utility

### GLOSSARY

=====

#### CREATE BIT

This refers to a flag contained in the directory entry for a file. When this bit is set, the operating system will be kept from deallocating any unused space at the end of the file when the file is closed. Such a file will never shrink in size. The file will remain as large as its largest allocation. The flag is normally set by the DOS CREATE command.

#### CYLINDER

A cylinder is a term which represents all like-numbered tracks on all surfaces of a disk. For example, on a two surface media, track zero of surface zero and track zero of surface one are grouped together into cylinder zero. Where a disk uses only a single surface, the terms cylinder and track both represent the same space and are thus equivalent.

#### EXTENT

An extent contains data on the allocation of disk space. Each extent can contain the allocation information for a maximum of 32 contiguous granules. The contents of the extent tells you what cylinder stores the first granule of the extent, what is the number of that granule, and how many contiguous granules are in use in the extent.

#### GRANULE

A group of sectors is allocated whenever additional space is needed for a file. This group is called a GRANULE and is always a constant size for any given disk. The size of a granule generally increases with the increasing size of the disk storage device.

#### K

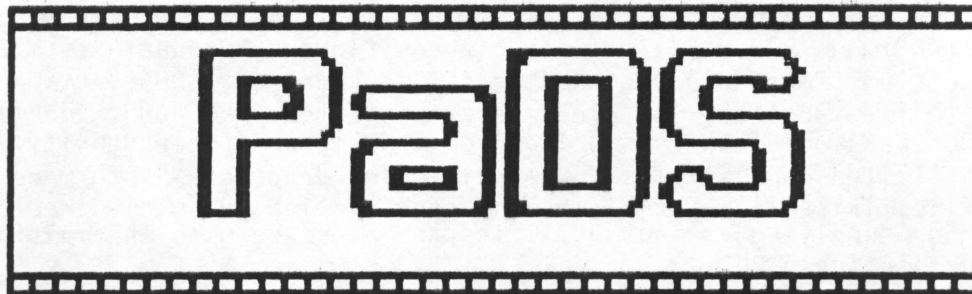
A "K" is a term applied to 1024 characters of storage. When applied to a disk, it would represent four sectors of space.

#### SECTOR

Each track of a disk is divided into subareas called sectors. Each sector has an identifying number within the track. In LDOS and TRSDOS systems, this sector usually holds 256 characters. The sectors are usually numbered starting from zero on each track.

#### TRACK

The magnetic layer of particles on a disk's surface are magnetized into concentric circles of storage during the formatting process. Each circle is called a track. A typical 40-track disk has a density of 48 tracks per inch. Thus the diskette region used has a lateral width of 0.833 inches.



## TABLE OF CONTENTS

=====

INTRODUCTION . . . . .	2
GENERAL INFORMATION . . . . .	3
COMMAND ENTRY . . . . .	5
PROCEDURES TO CREATE A PDS . . . . .	6
PATCH TO SYS0/SYS (5.0.2 & 5.0.3) . . . . .	7
PDS(APPEND) . . . . .	8
PDS(BUILD) . . . . .	12
PDS(COPY) . . . . .	14
PDS(DIR) . . . . .	16
PDS(KILL) . . . . .	18
PDS(LIST) . . . . .	20
PDS(PURGE) . . . . .	22
PDS(RESTORE) . . . . .	25

Authored and Copyrighted (C) 1982 Roy Soltoff  
Published by MISOSYS, PO Box 4848, Alexandria, VA 22303-0848.

LDOS is a trademark of Logical Systems, Inc.  
TRSDOS is a trademark of Tandy Corp.



## Partitioned Data Set Utility

### INTRODUCTION

=====

This material documents the implementation and use of the Partitioned Data Set (PDS and PRO-PaDS) Utility, a copyrighted product of MISOSYS. The PDS product functions solely with specific releases of LDOS Version 5 and is supplied on a 35-track single density diskette. PRO-PaDS functions with LDOS/TRSDOS Version 6 and is supplied on a 40-track single density diskette. PDS adds limited capabilities for partitioning data sets into more than one logical file. This is accomplished by constructing a mini-directory within the PDS and supplying various utilities to interface with the multiple member files constituting a PDS.

The PDS also contains a loader module that interfaces with the LDOS system loader so that executable members (/CMD type files) may be executed directly from "LDOS Ready" just as if they were regular files. Data file members are known by the loader and are inhibited from execution.

Various utility programs are supplied to support the management of the PDS. They perform specific functions such as directory display, member adding, killing, purging, copying, and appending. The entire PDS Utility package is itself supplied as a partitioned data set.

PDS was designed to offer LDOS users a capability of custom library generation. Additionally, it provides advantages of disk space compression when storing many small data files (or small /CMD files) in one PDS. This is particularly useful, as well as necessary, for the hard disk environment.

As more LDOS users make use of the PDS function, invariably there will be requests for additional capabilities. You are encouraged to make your requests known to MISOSYS. We will do our best to plan an orderly evolution of this product. To keep you directly informed of PDS enhancements and possible corrections, please take a moment and complete the warranty registration form and return it to MISOSYS. The registration number is located on the diskette label. Thank you and I wish that you receive splendid use of this product.

## Partitioned Data Set Utility

### GENERAL INFORMATION

=====

Partitioned Data Sets (PDS) are not new to LDOS. The two library files, SYS6/SYS and SYS7/SYS, are PDS structures [under Version 6, SYS8/SYS is also a partitioned data set]. Katzan, in OPERATING SYSTEMS, A PRAGMATIC APPROACH, defines a partitioned data set as "a data file that is divided into sequentially organized members." Katzan further states, "Each PDS includes a directory that points to the beginning of each member. Data sets of this type are most frequently used to store object programs - each member corresponds to a single object program. The PDS as a whole is referred to as a library. Operating system libraries and user libraries are stored in this fashion." This definition describes exactly, the two [or three] LIB files in LDOS.

The LIBRARY PARAMETER (LIBPARM) table located in SYS1/SYS contains two code values for each library command. One is a code which identifies the proper SYSn/SYS file containing the program which executes the command. The other code is an ISAM directory entry number used to identify its entry in the ISAM table directory located within the PDS SYS file itself. The LIBPARM table is used by the LDOS Command Interpreter which parses the command line and checks if the "filename" entered by the user matches up with a LIB command listed in the table. When a match is found, linkage is established with the system loader in SYSRES to denote the LIB file and the specific member entry satisfying the LDOS command entered. The member entry is denoted by the ISAM number assigned.

The system loader, opens the SYSn/SYS LIB file and reads through the ISAM directory table stored in the LIB file looking for a match to the ISAM entry number supplied by SYS1/SYS. This ISAM directory contains information on the position and transfer address for each member in the file. Once the system loader finds the appropriate entry, it positions the file to the starting point of the member then loads and executes the module.

The PDS structure has provided a technique for combining separately executable object programs into one file thereby saving directory slots. It also serves to save disk space because each member is concatenated without having to reside on a granule boundary. It also saves time by not having to load an entire 10K-15K file just to get a few hundred bytes or a few thousand bytes of program loaded if all LIB commands were just one big file. The overhead of having to read and search the member directory is minimal. This technique has been used for years on mainframe and mini computers.

The PDS utility functions with LDOS 5.1.x (Model I and Model III). PDS will function with earlier Model I versions 5.0.2 and 5.0.3 once a minor patch is applied to SYS0/SYS. This patch is identified as "SYSOPDSA/FIX" later in this documentation. The PRO-PaDS utility functions with LDOS/TRSDOS version 6.

The PDS command can be used to create custom user libraries. A library could be a collection of a dozen utility programs - all stored under one name but directly executable by specifying the library name followed by the member name. Consider for a moment, that you have built a library containing PROCESS, DSMBLR, BINHEX, EDAS, and XREF. The library name MYLIB was chosen. You can then execute EDAS by entering:

## Partitioned Data Set Utility

### MYLIB(EDAS)

at the "DOS Ready" prompt. If you wanted to build a custom DOS command library, you could use CMDFILE [Version 6 users could use PRO-CESS] to extract DIR, COPY, KILL, DEBUG, ROUTE, and RESET from SYS6/SYS and SYS7/SYS and build them into a user SYSLIB. Then you could purge SYS6/SYS and SYS7/SYS which would save about 15K-20K from your "custom" SYSTEM disk. When you want to do a directory, you would only need to type:

### SYSLIB(DIR) :2 (A,I)

to achieve the same result as if you had typed DIR :2 (A,I) on a regular SYSTEM disk. Albeit you could have named your user library, "S" and save the entering of five characters each time you wanted to execute a member of the library. That would let you use "S(DIR)"!

Okay, what capabilities are included with PDS and PRO-PaDS? The PDS command is itself a Partitioned Data Set. Members provided implement the following functions:

**APPEND** - Appends a new member or members to the existing PDS and updates the member directory and map tables accordingly. Executable program members may have more than one entry point when appended with the MAP parameter.

**BUILD** - Provides the capability of creating a new partitioned data set with a user designated maximum number of members. The PDS is composed of a Front End Loader program, a MEMBER directory, and an ISAM directory MAP table. Members are added via the PDS(APPEND) command.

**COPY** - Transfers an image of a PDS member from the PDS to a designated file. The member will not be deleted from the PDS.

**DIR** - Provides a directory of members listing the member name, member type, date of addition, and file space occupied.

**KILL** - Makes a member inaccessible for access.

**LIST** - Will list a specific member in standard hex format or ASCII format.

**PURGE** - Removes killed member(s) from the PDS and compresses the file to reclaim the space previously occupied by the killed member(s).

**RESTORE** - Restores a killed file to accessibility.



## Partitioned Data Set Utility

### COMMAND ENTRY

=====

The complete file specification for use with PDS files is:

FILENAME/EXT.PASSWORD:DRIVE(MEMBERSPEC)

```
*****
*
* A word of caution. The MEMBER specification is valid only when
* used with a PDS command. You cannot provide the memberspec in
* a file specification used with LDOS commands not specifically
* designed to function with Partitioned Data Set files. Use the
* the memberspec only with the PDS command library!
*
*****
```

Any PDS executable program member may, of course, be executed directly at "DOS Ready" or via the RUN library command as well as from LBASIC's CMD"command" in Version 5 or BASIC's SYSTEM"command" in Version 6. When entering a PDS file specification for execution, any MEMBER name can be shortened to its minimum length necessary to uniquely match up with the member directory. For example, each PDS library command begins with a distinct letter of the alphabet; therefore, all PDS library commands can be abbreviated to a single character and still retain their uniqueness. If two members were named COPY and CREATE, then the minimum length necessary to uniquely identify either would be a length of two [CO, CR].

Where member names are used within PDS commands, such as PDS(KILL), or PDS(COPY), then the entire memberspec must be entered. The command line entry abbreviation was chosen to minimize keystrokes where member names are frequently entered. The full memberspec is required within PDS commands to maintain maximum integrity of the PDS file and associated maintenance functions. Remember, a PDS may contain many members. Therefore, additional measures of protection are vital to ensure the integrity of the file.

## Partitioned Data Set Utility

### PROCEDURES TO CREATE A PDS

=====

The creation of a Partitioned Data Set is an easy task. To begin with, you should ask yourself what is it to be used for? Perhaps a collection of scores of utility programs - all combined into one larger file. A larger file will undoubtedly save disk space since small files will always use at least one granule of space [1.25K in 5-1/4" SDEN, 1.5K in 5-1/4" DDEN, 4K in 5-1/4" rigid, ...]. Members of a PDS use only that amount of space necessary to contain each member. The entire PDS file uses space rounded to integral granules. So it makes good sense to concatenate your small /CMD files into one or more PDS files. How many will you want to store? Use that estimate, plus a few extra to handle those you forgot about, as the MEMBERS parameter in the PDS(BUILD) command to initialize a new file for use as a Partitioned Data Set.

Second, concatenate the /CMD files by using the PDS(APPEND) command. You could, of course, build a MAP file listing each addition to be made and use the MAP option of APPEND to streamline the process. Or use JCL to PDS(APPEND)! Then use the PDS - it's simple to deal with.

Perhaps you would really like to be able to store a large number of small data files on a disk. Assume that you are using your machine for word processing in an environment where it is important to catalog letter files. Letters typically waste two to four sectors of storage space because file allocations are always made in granules. That can mean 500 to 1000 characters of storage space lost for each letter which can use up disk space fast. PDS(BUILD) a Partitioned Data Set to store 50-100 letters - each having a unique eight-character member name with their own mini-directory. The PDS file name can then become another field for use in cataloging the letters. PDS(BUILD) a couple of PDS "letter" files for specific categories. Then a convention can be established for the member names that will assist in denoting their contents. Remember, the PDS(DIR) command produces a sorted directory to aid you. You will also be optimizing the storage space available on your "letter" disks providing storage space for more letters per diskette.

The following example will illustrate the steps required to initialize a ten-member Partitioned Data Set and concatenate five members - four of which are executable programs and one of which is a pure text documentation file.

PDS(BUILD) ULIB:2 (M=10)	;Creates the 10-member PDS
PDS(APPEND) DSMBLR ULIB.PDS:2	;Appends DSMBLR/CMD
PDS(APPEN) XREF ULIB.PDS:2	;Appends XREF/CMD
PDS(APPE) PARMDIR ULIB.PDS:2	;Appends PARMDIR/CMD
PDS(APP) CONVCPM ULIB.PDS:2	;Appends CONVCPM/CMD
PDS(A) NOTE/SCR:3 ULIB.PDS:2	;Appends NOTE/SCR

Note that the member name, APPEND, can be abbreviated to as few a quantity of characters that still keep its name unique amongst all the members in the PDS! The example illustrated this convenience by denoting various acceptable forms for entering the member name. You can now execute DSMBLR, for instance, by entering:

ULIB(D) or ULIB(DS) or ULIB(DSM) or ULIB(D

## Partitioned Data Set Utility

The closing parenthesis following the member name is required only when additional information is entered on the command line. Again note that the member name, DSMBLR, can be abbreviated to a single character because no other member name starts with the letter "D".

### PATCH TO SYS0/SYS (5.0.2 & 5.0.3)

=====

If you are going to use PDS with LDOS Model I Version 5.0.2 or Version 5.0.3, the following patch must be implemented prior to using the PDS utility. DO NOT APPLY THIS PATCH TO ANY OTHER VERSION OF LDOS.

- . SYSOPDSA/FIX - 12/20/81 - by Roy Soltoff
- . This patch is for LDOS 5.0.2 or LDOS 5.0.3 only!
- . PATCH SYS0/SYS.WOLVES

.  
DOC,35=11 FF 42 CD 0A 4D DD E1 C9  
. WAS CD 07 4D DD E1 C9 11 FF 42  
. End of patch



## Partitioned Data Set Utility

### PDS(APPEND)

=====

This PDS library command is used to add new members to an existing Partitioned Data Set (PDS). The syntax is:

```
=====
|
| PDS(APPEND) filespec1 filespec2 (MAP,DATA)
| PDS(A) filespec1 filespec2 (MAP,DATA)
|
| filespec1  Is the file you wish to add as a MEMBER, or
|             is the name of a MAP file which contains the
|             the filespecs of the files to add.
|
| filespec2  Is the PDS file to receive the MEMBER.
|
| MAP        Indicates that filespec1 is a MAP file.
|
| DATA      Forces the added MEMBER to be interpreted as
|             a data file [in lieu of a program file].
|
| abbr: MAP=M, DATA=D
|
|=====
```

The APPEND command can be used to add either one or more members to the PDS identified as "filespec2". Members may be executable programs [/CMD type files] or data files [anything which is not a /CMD type file]. Any appending file with a file extension of "/CMD" which has as its first byte either an X'01' or an X'05' and has as its fourth from last byte an X'02' will be interpreted as an executable program file unless overridden by the DATA parameter. Any appending file not so identified will be interpreted as a data file. Since executable program files have the X'02' byte changed to an X'04' when stored in the partitioned data set, the DATA parm is supplied to restrict PDS(APPEND) from treating a data file, which matches the program file specification, as a program file. It should be quite rare to discover a data file which matches the executable program parameters.

You may discover that some /CMD files in your possession may not have a proper end-of-file value in the directory. You can ascertain if a file is properly registered in a directory by listing the file in hex with the DOS's LIST command. A properly generated /CMD file will end in "02 02 TL TH", where TL and TH are the low order and high order bytes of the module's transfer address. If you PDS(APPEND) a /CMD file that is improperly registered, it will be assumed to be a data file. Thus, an attempt to execute it from the PDS would result in a "load file format error" message from LDOS. In order to correct such a file, you can use a utility that reads a load module file until the transfer address record while it writes out to another file only that portion read. The CMDFILE or PROCESS utilities may be used for this purpose.

A data file member cannot be executed by the operating system. Any attempt to do so will result in an immediate abort with the "Load module

## Partitioned Data Set Utility

format error" message being issued. Data file members can be catalogued for archival or other purposes and retrieved by the PDS(COPY) command.

A program member can have one or more entry points, each entry point identifiable as a distinct MEMBER name. Each entry point will require a separate PDS directory entry; however, the module is stored only once in the PDS. If multiple entry points are required, then you must prepare a MAP file which is used to append the member to the PDS. The MAP file data line provides the means by which multiple member names and entry points are identified to the PDS(APPEND) command. It is a good idea to use a file extension of "/MAP" on a MAP data file. The reason will soon become evident. The format of a MAP file is as follows:

```
filespec1,member1,traadr1,member2,traadr2,...  
filespec2,member1,traadr1,member2,traadr2,member3,traadr3,...  
filespec3,member1,traadr1,...  
.  
.  
.
```

As can be noted, each line contains the information for a single file which can have one or more member names and transfer addresses. The line is limited to 63 characters in length. Each line must be terminated with an <ENTER>. The comma "," is used to separate each field on a line. Also, spaces cannot appear in the line.

The first field specifies the name of the file that is to be loaded. "Member1" and "traadr1" is the first member name and its entry point. Subsequent fields identify each member name (another entry to filespec) and the respective entry point. For example, suppose you had a file called, MOVE/CMD, which contained two entry points: APPEND at X'5200' and COPY at X'5203'. The MAP record to append such a file would be:

```
MOVE,APPEND,5200,COPY,5203
```

If you are using the PDS file to store assembler libraries and you have multiple entry points to an assembler source code member, what do you do about the transfer address? There is no such thing for the source! Well, since the syntax requires a transfer address, you must enter something; however, it doesn't matter what is entered - a zero "0" will suffice.

So far, the MAP parameter was used to specify multiple entry points. It has two other purposes. First, instead of a second or third entry point, "member2" could be an "alias" with the same entry point. Thus, the same member could be accessible via two different names. Second, where you have a large number of members to add at one time, identifying each entry as a MAP record and using the "PDS(APPEND) filespec/MAP pdsfilespec (MAP)" method will result in appending the members more quickly.

Both filespec1 and filespec2 will have their file extensions default to "/CMD". Therefore, if no extension is provided, "/CMD" will be assumed. If the MAP parameter is entered, then the default file extension for filespec1 will be "/MAP". This guards against inadvertently appending a MAP file to a PDS by forgetting to enter the MAP parameter. This is why it was recommended

## Partitioned Data Set Utility

that "/MAP" be used as the file extension for the MAP file.

### Example

-----

PDS(APPEND) DSMBLR S.PDS:0

will add the program, DSMBLR/CMD, to the "S/CMD.PDS:0" partitioned data set.

### Informative Messages

-----

#### READING PDS MEMBER DIRECTORY...

This message will be displayed after the PDS file has been opened and when the PDS member directory is loaded into memory.

#### READING filespec TO APPEND...

This message will be displayed when the file to be added as a member is being read into memory.

#### APPENDING FILE TO PDS...

This will be displayed when the new member is written to the PDS.

#### UPDATING PDS MEMBER DIRECTORY...

This message will be displayed when the PDS directory is being corrected to contain the information on the new member.

#### MEMBER {memberspec} ADDED TO PDS.

This message will be displayed upon successful completion of the member addition to the PDS. The member name will be shown for "memberspec".

### Error Messages

-----

#### INPUT FILESPEC REQUIRED!

This error indicates that the file specification for the appending file was not entered on the command line.

#### PDS FILESPEC REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.



## Partitioned Data Set Utility

### PARAMETER ERROR!

An error was encountered in one or more parameters passed in the parameter string.

### DESTINATION FILE IS NOT A PDS!

The filespec entered for the PDS did not define a partitioned data set file. A DOS directory command displays an asterisk adjacent to PDS files.

### PDS DIRECTORY IS FULL!

There is no more room in the PDS directory to add another member. Either start a new PDS or PDS(PURGE) unwanted existing members.

### MEMBER {memberspec} ALREADY IN PDS!

The PDS already contains a member with the same name as the appending file's FILENAME. The new member will not be added.

### OUT OF MEMORY - INPUT FILE TOO BIG!

The appending file must be small enough to fit into available memory. The file is probably too big to be considered useful for membership in a PDS. You may also have too little memory available due to excessive high-memory usage (spool buffers, SYSRES modules, filters, etc.).

### MAP INPUT FORMAT ERROR!

A syntax error was detected in a line of the MAP file. The offending line up to the character considered to be in error is displayed. Check your MAP file for adherence to the documented construct.

### MAP INPUT MEMBER NAME ERROR!

Most likely, the member name field was longer than the maximum eight characters allowed. The offending line up to the erroneous character will be displayed.

### MAP INPUT DECK NAME TOO LONG!

This error will be displayed if the file specification field of the MAP record contains a file name that exceeds eight characters in length. The offending line up to the erroneous character will be displayed.

## Partitioned Data Set Utility

### PDS(BUILD)

=====

This PDS library command is used to initialize a file for use as a partitioned data set. The syntax is:

```
=====
|
| PDS(BUILD) filespec (MEMBERS=ddd,LOADER="filespec")
| PDS(B) filespec (MEMBERS=ddd,LOADER="filespec")
|
| filespec  Is the partitioned data set to be initialized.
|           It will be created if non existing.
|
| MEMBERS   Is the directory size in members. The PDS will
|           be created with a size of 16 if omitted. The
|           parameter value must be in the range <1-255>.
|
| LOADER    Is used to initialize the PDS with a Front End
|           Loader (FEL) module other than the standard
|           FEL supplied with the PDS utility.
|
| Abbr: MEMBERS=M [PRO-PaDS accepts LOADER=L]
|
=====
```

The PDS(BUILD) command must be used to initialize a partitioned data set prior to adding the first member with the PDS(APPEND) command. If "filespec" is an existing PDS file and the password is included, it will be overwritten with an empty directory. A Front End Loader program as well as empty MEMBER and ISAM tables constituting the directory, are used to initialize the file for use as a partitioned data set.

A PDS is an extremely important file since it can contain up to 255 members and each member is itself a file. It is essential, therefore, that you do not inadvertently kill, write to, or copy to a partitioned data set. In fact, the only time writing should be done to a PDS is during a member addition via the PDS(APPEND) command and certain other PDS commands. For this reason, BUILD will automatically assign password protection to a Partitioned Data Set during the building process by invoking the DOS ATTRIB command. The following attributes will be assigned:

-----LDOS Version 5-----  
UPDATE PASSWORD: PDS  
ACCESS PASSWORD: blanks  
PROTECTION LEVEL: READ

-----LDOS Version 6-----  
OWNER Passwor : PDS  
USER Password: blanks  
Protection Level: READ

You may change the password to one of your selection by using the system ATTRIB command after the PDS is built. It is strongly suggested that you NOT remove the protection so as to avoid inadvertant destruction of the PDS. Remember, most of the accesses to a Partitioned Data Set will be in a READ mode and will not require entry of the password. The password would only be needed to append a member, KILL a member, PURGE all KILLED members, rename the entire PDS, or KILL/REMOVE the entire PDS.

## Partitioned Data Set Utility

A Partitioned Data Set can contain a maximum of 255 members - the maximum sized directory. The maximum size is defined at initialization. If you do not enter a MEMBERS parameter, a value of 16 directory slots will be provided. A directory size chosen should depend on your drive capacity and file sizes to be included as members.

In order to provide automatic execution of an executable program file member, a Front End Loader (FEL) program is required in every Partitioned Data Set. A standard FEL is supplied within the PDS(BUILD) library command that supports the execution of program members as identified in this document. It is automatically used unless overridden by the LOADER parameter for custom PDS implementations. Use of the LOADER parameter will result in the prompting of the loader file specification. If desired, the loader file specification may be entered with the parameter word in the format:

**LOADER="filespec"**

If the file specification is not entered with the parameter, it will be prompted for. It is beyond the scope of this documentation to provide technical information concerning Front End Loaders.

### Informative Messages

-----

**WHAT IS THE NAME OF YOUR LOADER? >**

This prompt message will be displayed if you enter the LOADER parameter without specifying the file specification in the command line. Enter the filespec in response to the prompt.

**PDS FILE GENERATION COMPLETE**

This message indicates successful completion of the PDS initialization.

### Error Messages

-----

**PDS FILE SPECIFICATION REQUIRED!**

This error indicates that the Partitioned Data Set filespec was omitted from the command line.

**PARAMETER ERROR!**

An error was encountered in one or more parameters passed in the parameter string.

**MANUAL BREAK ABORT!**

The BREAK key was depressed in response to a request for the loader filespec. The BUILD process terminates.



## Partitioned Data Set Utility

### PDS(COPY)

=====

This PDS library command will copy a MEMBER from a partitioned data set (PDS) to a standard type file. It can be used to retrieve archived member data files. The syntax is:

```
=====
|
| PDS(COPY) filespec1(memberspec) filespec2
| PDS(C) filespec1(memberspec) filespec2
|
| There are no parameters
|
=====
```

This command will transfer the image of a member identified as "memberspec" from the partitioned data set identified as "filespec1" to a file identified as "filespec2". Both filespecs will be assumed to have a file extension of "/CMD" unless one is provided.

If the member being copied was an executable program having multiple entry points, the transfer address provided for the destination file will be the entry point of the member name provided in the command line.

### Example

-----

```
PDS(COPY) MYLIB:5(EDAS) EDAS:1
```

will copy the member, EDAS, from the partitioned data set, MYLIB/CMD:5, to a file named EDAS/CMD:1

### Informative Messages

-----

COPYING MEMBER memberspec TO FILESPEC

This message will be displayed while the member is being copied.

COPY FUNCTION COMPLETE

This message will be displayed at the conclusion of the copy process.

### Error Messages

-----

PDS FILESPEC REQUIRED!

This error indicates that the Partitioned Data Set filespec was omitted from the command line.

## Partitioned Data Set Utility

### PDS MEMBER REQUIRED!

The memberspec was omitted from the PDS file specification.

### MEMBER NAME ERROR!

Most likely the memberspec exceeded the eight-character maximum length.

### SOURCE FILE IS NOT A PDS!

The file identified by the PDS file specification is NOT a partitioned data set.

### MEMBER NOT IN PDS DIRECTORY!

The member identified in the PDS file specification was not found in the PDS directory.

### DESTINATION SPEC REQUIRED!

A file specification satisfying the destination file was omitted from the command line.

### OUT OF MEMORY - MEMBER FILE TOO BIG!

The member file must be small enough to fit into available memory. Since sufficient memory was available to add the member to the PDS, you may have too little memory currently available due to excessive high-memory usage (spool buffers, SYSRES modules, filters, etc.).

## Partitioned Data Set Utility

### PDS(DIR)

=====

This PDS library command is used to obtain a directory of partitioned data set MEMBERS. The syntax is:

```
=====
|
| PDS(DIR) filespec (K,P)
| PDS(D) filespec (K,P)
|
| filespec  Is the PDS file whose directory is listed.
|
| K        Is entered to list the member names of killed
|          members in addition to the active members.
|
| P        Is entered to simultaneously direct the output to
|          the *PR device.
|
|=====
```

The PDS(DIR) command will provide an alphabetized listing of all active and killed members in a PDS. In addition to the member name, the listing will show either a "P" or "D" to indicate a program or data file member, the date that the member was added to the PDS, and the length of the member in bytes. If the "K" parameter is entered, then killed members will be listed and noted by the presence of an asterisk following the member name. All active members will be listed first followed by the killed members.

### Example

-----

#### PDS(DIR) PDS

PDS: PDS/CMD	08/11/83	Size:	9K	Members:	8/ 10
append	P 15-Mar-83 1703	build	P 15-Mar-83	1100	
copy	P 15-Mar-83 1041	dir	P 15-Mar-83	1288	
kill	P 15-Mar-83 551	list	P 15-Mar-83	1042	
purge	P 15-Mar-83 1464	restore	P 15-Mar-83	601	

### Informative Messages

-----

There are no informative messages other than the listing.



## Partitioned Data Set Utility

### Error Messages

-----

#### PDS FILE SPECIFICATION REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.

#### PARAMETER ERROR!

An error was encountered in one or more parameters passed in the parameter string.

#### FILE IS NOT A PDS!

The file specification entered for the PDS did not define a partitioned data set file.

## Partitioned Data Set Utility

### PDS(KILL)

=====

This PDS library command is used to remove a PDS member from the PDS directory. The member is left intact until purged. The syntax is:

```
=====
| PDS(KILL) filespec(membername)
| PDS(K) filespec(membername)
|
| no parameters are required
|
=====
```

This command can be used to remove a member from the PDS directory. Members that are killed cannot be executed, listed, copied or otherwise accessed. Killed members will also not appear in a directory listing unless the "K" parameter is provided. A killed member is denoted in the directory by having the member name's first character high-order bit set to a one. Killed members may be restored to active status with the PDS(RESTORE) command unless the member has been purged.

### Example

-----

PDS(KILL) MYLIB.PDS:1(JUNKFILE)

will make inactive the member, JUNKFILE, in the Partitioned Data Set, MYLIB/CMD.PDS:1.

### Informative Messages

-----

KILL FUNCTION COMPLETE.

This is displayed upon successful completion of the PDS(KILL) function.

### Error Messages

-----

PDS FILESPEC REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.

PDS MEMBER REQUIRED!

The member specification was omitted from the PDS file specification.

## Partitioned Data Set Utility

### MEMBER NAME ERROR!

Most likely the memberspec exceeded the eight-character maximum length.

### SOURCE FILE IS NOT A PDS!

The file identified by the PDS file specification is NOT a partitioned data set.

### MEMBER NOT IN PDS DIRECTORY!

The member identified in the PDS file specification was not found in the PDS directory.



## Partitioned Data Set Utility

### PDS(LIST)

=====

This PDS library command can be used to display a Partitioned Data Set MEMBER in either ASCII mode or HEX mode in a manner similar to the system's LIST command. The syntax is:

```
=====
PDS(LIST) filespec(memberspec) (ASCII,PRINT)
PDS(L) filespec(memberspec) (ASCII,PRINT)

filespec(memberspec) Is the PDS member to be listed.

ASCII      Indicates an ASCII listing is desired.

PRINT      Denotes that the output is to simultaneously
            be directed to the *PR device.

Abbr: ASCII=A, PRINT=P
=====
```

A Partitioned Data Set MEMBER can be listed to the \*DO device and the \*PR device by using this PDS library command. The listing will normally be in standard HEX display unless the parameter, "ASCII", is entered. The ASCII display will not be line numbered.

The listing is normally presented to the \*DO device. If you want to direct the output to a line printer, the PRINT parameter can be entered.

During the listing, depression of the BREAK key will abort the operation and return you to the "DOS Ready" prompt. The PAUSE key [usually activated by simultaneous depression of the SHIFT and @ keys] will temporarily suspend the listing. Depression of any other key will resume the listing.

### Example

PDS(LIST) LETTERS/SCR:5(L1221005) (A)

will list the member, L1221005, from the LETTERS/SCR:5 Partitioned Data Set. The listing will be in ASCII.

### Informative Messages

-----

There are no informative messages.

## Partitioned Data Set Utility

### Error Messages

-----

#### PDS FILESPEC REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.

#### PARAMETER ERROR!

An error was encountered in one or more parameters passed in the parameter string.

#### PDS MEMBER REQUIRED!

The member specification was omitted from the PDS file specification.

#### MEMBER NAME ERROR!

Most likely the memberspec exceeded the eight-character maximum length.

#### SOURCE FILE IS NOT A PDS!

The file identified by the PDS file specification is NOT a partitioned data set.

#### MEMBER NOT IN PDS DIRECTORY!

The member identified in the PDS file specification was not found in the PDS directory.

## Partitioned Data Set Utility

### PDS(PURGE)

=====

The PDS PURGE library command will physically remove killed members from a Partitioned Data Set. The storage space occupied by the purged members will be reclaimed. The syntax is:

```
=====
|
| PDS(PURGE) filespec (QUERY=N)
| PDS(P) filespec (Q=N)
|
| filespec  Is the PDS file to purge.
|
| QUERY=N   Indicates that all killed members are to be
|           purged without prompting.
|
| Abbr: QUERY=Q
|
=====
```

This PDS library command is used to eliminate space in a Partitioned Data Set occupied by "killed" members. The member name, type of member, and append date of each killed member will be identified to confirm its purge. You thus have the opportunity to selectively purge killed members. If the parameter, "QUERY=N", is entered, then all killed members will be purged without the display of prompting messages.

Purging is a multi-step process. In the first step, the name of each killed member is displayed. You have the option of entering "Y" to purge it, or "N" to not purge it. If you enter a BREAK, the purge process will abort.

In the second step, a member will be purged by bubbling up all members in the PDS that follow the killed member in storage. This will overwrite the killed member with successively stored members. The directory will be updated to reflect the revised pointers to all bubbled up members. The process will be repeated for all members requested to be purged. Since it is possible by using MAP appending to have multiple entry points to a member (it results in more than one member name for a physical member), the PDS(PURGE) process will automatically purge all entry points to a PDS member regardless if they are all killed or not. Therefore, it is not necessary for you to kill all the member entry names to a member in order to purge it. It also causes no harm if more than one member name of a multiple entry member is killed. All member names that are purged will be displayed.

In the final step, the PDS directory is written back to the PDS and the system directory is updated.

The larger the PDS, the greater the disk I/O needed to accomplish the purge operation. If your system frequently has disk errors, you may want to avoid purging killed members. It is recommended that you make a copy of the PDS prior to the purge operation in case of error. Purging is not absolutely essential. The same process could be achieved by a series of PDS(COPY) and



## Partitioned Data Set Utility

PDS(APPEND) commands by transferring active members to a new Partitioned Data Set.

### Informative Messages

-----

#### READING PDS MEMBER DIRECTORY...

This message will be displayed after the PDS file has been opened and when the PDS member directory is loaded into memory.

#### PURGE MEMBER: memberspec P/D date ?

This message will be displayed for each killed member in the PDS (assuming Q=Y). Answer Y or N.

#### BEGIN PURGING <Y,N>? >

This is the last message displayed before file I/O commences. If you want to abort the purge, this is your last chance.

#### MEMBER {memberspec} PURGED FROM PDS.

This will be displayed after a member is purged and the PDS directory is updated in memory. The PDS directory has not yet been written to the disk.

#### UPDATING PDS MEMBER DIRECTORY...

This message will be displayed when the PDS directory is being corrected on disk to contain the revised directory information.

### Error Messages

-----

#### PDS FILESPEC REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.

#### PARAMETER ERROR!

An error was encountered in one or more parameters passed in the parameter string.

#### FILE IS NOT A PDS!

The file specification entered for the PDS did not define a partitioned data set file.

## Partitioned Data Set Utility

### PURGE ABORTED!

The **BREAK** key was depressed in response to informative messages which required an entry. The PURGE process terminates.

### NOTHING TO PURGE!

Either the file contained no killed members or none of the displayed killed members were requested to be purged.

### UNEXPECTED ERROR - {memberspec}: RECOVERY BEING ATTEMPTED!

What can be said? Any number of conditions could cause this error message. Any file positioning error prior to a new I/O sequence will abort the purge and proceed to write the PDS directory back to the disk - hopefully maintaining a usable PDS. Also, an error detected in the PDS directory records will also produce this error message. Any disk I/O error will abort the purge with the PDS file's integrity in jeopardy. This is the most serious message to obtain. A backed up copy of your PDS is your insurance.

## Partitioned Data Set Utility

### PDS(RESTORE)

=====

This PDS command will restore a killed member that has not been purged.  
The syntax is:

```
=====
|
| PDS(RESTORE) filespec(memberspec)
| PDS(R) filespec(memberspec)
|
| filespec(memberspec) Is the PDS member to be restored.
|
| There are no parameters
|
=====
```

This PDS library command will restore a member that has been killed from a Partitioned Data Set. If the member had been purged, no amount of "restoral" will reclaim the member.

### Example

-----

PDS(R) MYLIB(IGOOFED)

will restore the member, IGOOFED, in the Partitioned Data Set designated as "MYLIB/CMD".

### Informative Messages

-----

RESTORE FUNCTION COMPLETE.

This will be displayed upon successful restoral of the member and its return to active status.

### Error Messages

-----

PDS FILE SPECIFICATION REQUIRED!

This error indicates that the Partitioned Data Set file specification was omitted from the command line.

PDS MEMBER REQUIRED!

The member specification was omitted from the PDS file specification.



## Partitioned Data Set Utility

### MEMBER NAME ERROR!

Most likely the memberspec exceeded the eight-character maximum length.

### SOURCE FILE IS NOT A PDS!

The file identified by the PDS file specification is NOT a partitioned data set.

### MEMBER NOT IN PDS DIRECTORY!

The member identified in the PDS file specification was not found in the PDS directory.

### MEMBER IS NOT KILLED!

The member you requested to have restored was not a "killed" member.

## X-FTS - X-Modem protocol File Transmission System

### TABLE OF CONTENTS

General Information . . . . .	1
Invoking X-FTS . . . . .	2
USING X-FTS . . . . .	3
X-FTS PARAMETERS . . . . .	5
RS-232 PARAMETER SETTINGS . . . . .	7
JCL EXECUTION OF X-FTS . . . . .	7
USING FILTERS ON THE RS-232 DEVICE . . . . .	7
OPERATING SYSTEM PECULIARITIES . . . . .	7

X-FTS/CMD: Copyright 1984 by Rick C. Francis, All rights reserved. X-FTS is published by MISOSYS, Inc., Sterling VA 22170.

LDOS and LS-DOS are trademarks of Logical Systems Inc.  
TRSDOS is a trademark of Tandy Corp.  
DOS PLUS is a trademark of Micro Systems Software

### GENERAL INFORMATION

This documentation covers the Model I/III version of X-FTS which functions under the LDOS 5.1 operating system. It also covers the TRSDOS 6.x or LS-DOS 6.x version of X-FTS called PRO-X-FTS. The PRO version also operates under DOS PLUS IV. The specific version of X-FTS is noted on the diskette label supplied with this package. X-FTS provides the user with a communications tool to transmit and receive disk files via an industry standard protocol which effectively supports error-free transmission.

## X-FTS - X-Modem protocol File Transmission System

### Invoking X-FTS

X-FTS is a general purpose file transmission utility for use with LS-DOS 6.X, TRSDOS 6.X and DOS PLUS IV (PRO-X-FTS), or LDOS 5.1 (X-FTS). It allows you to send any file to another computer error-free via the RS-232. The program is compatible with the Christensen or XMODEM protocol which is very popular with CP/M and MS-DOS users. The syntax is:

```
FTS x filespec (parameters)
FTS *x filespec (parameters)
```

```
x          - S to send or R to receive
*S or *R   - Forces JCL mode of execution
```

```
filespec   - Name of file to send or receive
```

Parameters:

```
ABS=sw     - If specified in receive mode,
              existing files will be replaced
              without prompting. Default is OFF.
```

```
BLOCK=b    - Specifies the size (in bytes) of
              the data blocks. Default is 128.
```

```
DEVICE=s    - Specify alternate RS-232 device.
              Default is "CL".
```

```
EOF=sw      - If specified in send mode, the
              end-of-file will be properly
              maintained to the byte level.
              Default is OFF.
```

```
FAST=sw     - Turns interrupts off during
              block receive for high baud rates.
              Default is OFF.
```

```
KEY=b       - Specifies a data encryption key.
              Default is 0 (no encryption).
```

```
LRL=b       - In receive mode this value will
              be used as the Logical Record
              Length for new files.
              Default is 0 (256 byte LRL).
```

Parameters continued on next page.



## X-FTS - X-Modem protocol File Transmission System

**NOTIFY=b** - Specifies the number of audible beeps to sound when the program terminates. Default is 0. If given without a value, 3 is used.

**QUIET=sw** - If specified, nothing will be on the screen during file transfer. Default is OFF.

**RETRY=b** - Specifies the maximum number of retries allowed on one block. Default is 9.

Abbr: All parameters except ABS may be abbreviated to one letter.

### Notes:

- "sw" is a switch value (ON or OFF).
- "b" is a byte value in the range 0 to 255.
- "s" is a string (i.e. "CL" or "\*CL")
- NOTIFY not supported under LDOS 5.1
- ABS and LRL valid in receive mode only.
- EOF valid in send mode only.
- For Christensen protocol, use default values for BLOCK and EOF.

## USING X-FTS

X-FTS may be used to transfer any file to another computer provided that the other computer is running X-FTS or a compatible program. We will refer to your computer as the "local" computer and the other computer as the "remote" computer. The two computers may be side-by-side connected directly to each other with an RS-232 cable or they may be hundreds of miles apart connected by telephone lines. For simplicity, we'll assume that both computers are running TRSDOS 6.2 with X-FTS and that they are connected with 300 baud modems. Suppose we want to send the file "MYFILE/DAT" from the local computer to the remote computer:

1. LOCAL: SET \*CL COM <ENTER>  
REMOTE: SET \*CL COM <ENTER>
2. LOCAL: SETCOM (B=300,P=OFF,S=1,W=8) <ENTER>  
REMOTE: SETCOM (B=300,P=OFF,S=1,W=8) <ENTER>
4. LOCAL: COMM \*CL <ENTER>  
REMOTE: LINK \*KI \*CL <ENTER>  
and LINK \*DO \*CL <ENTER>

## X-FTS - X-Modem protocol File Transmission System

The remote computer is now set up as a "HOST" computer and the local computer acts as a terminal to the remote. The local user can now give commands to the remote computer and see the results as though he were using the remote computer directly. From this point on, the transfer of files can be totally controlled by the local user. The following commands are issued by the local user:

4. FTS R MYFILE/DAT (QUIET) <ENTER>
5. <CLEAR> <SHIFT> <0>  
(COMM function to enter system command)
6. FTS S MYFILE/DAT <ENTER>

The file transfer will now take place. During the transfer, the local computer's screen will show the number of each block as it is transferred. It will also indicate the number of errors detected during the transfer. Each time an error is detected, the block is re-transmitted until the block is sent error-free. When the file transfer is complete, the local computer will be returned to the COMM program, and the remote computer will return to TRSDOS Ready.

Under LDOS 5.1, the command sequence above would be as follows:

1. LOCAL: SET \*CL RS232x (B=300,W=8,P=OFF,S=1) <ENTER>  
REMOTE: SET \*CL RS232x (B=300,W=8,P=OFF,S=1) <ENTER>
2. LOCAL: LCOMM \*CL <ENTER>  
REMOTE: Same as step 3 above.

The local computer now controls the transfer of files.

3. FTS R MYFILE/DAT (QUIET) <ENTER>
4. <CLEAR> <SHIFT> <=>  
(Exit LCOMM to LDOS Ready)
5. FTS S MYFILE/DAT <ENTER>
6. LCOMM \*CL <ENTER>

In step 1, RS232x should be replaced with the RS-232 driver name appropriate for your computer (RS232T, RS232R, RS232M, or RS232L for the Model III, Model I R/S interface, MAX-80, or Model I LX-80 interface, respectively).

## X-FTS - X-Modem protocol File Transmission System

### X-FTS PARAMETERS

#### ABS

When receiving files with X-FTS, this parameter may be used to force X-FTS to replace an existing file without prompting the user. The ABS parameter defaults to "OFF" unless X-FTS is running in JCL mode in which case it defaults to "ON". When sending files, this parameter has no effect.

#### BLOCK

The BLOCK parameter allows the user to set the size, in bytes, of the data blocks. The sending and receiving side must use the same value for this parameter. The default block size is 128. This is the block size used by the XMODEM protocol.

#### DEVICE

X-FTS normally uses the device "\*CL" for RS-232 communications. If you use a different device name for the RS-232, you must specify this with the DEVICE parameter. Under DOS PLUS IV, the RS-232 device is defined as "@RS" and the DEVICE parameter has no effect.

#### EOF

The CP/M operating system does not maintain its end-of-file marker down to the byte level. CP/M files are always some integer multiple of 128 bytes. Usually, a CONTROL-Z is used to mark the end of a text file but binary files have no byte-level EOF marker. The XMODEM protocol was first designed for CP/M systems so it provides no method of maintaining the EOF to the byte level. Under the TRSDOS family of operating systems, byte level EOF is maintained in the directory. By using the EOF parameter on the sending side, X-FTS will maintain the EOF marker during the transfer. EOF mode detection is automatic on the receiving side. The EOF parameter defaults to "OFF".

#### FAST

During block transfer, X-FTS normally allows interrupts to occur, this keeps the system clock accurate and allows for BREAK key detection. For high speed transfers, the RTC interrupt can cause characters coming from the RS-232 to be lost. Use the FAST parameter to cause X-FTS to disable interrupts during block receive. CAUTION: using the FAST parameter, will cause the system clock to lose time and will make BREAK key detection sluggish. The FAST parameter has no effect in send mode. Note: Due to the nature of the RS-232 driver for the Model III and MAX-80 (under LDOS 5.1), the FAST parameter must NOT be used. These drivers, with their large (128 byte) interrupt driven RS-232 buffer, will have no problem with high-speed transfers (9600 baud or less).



## X-FTS - X-Modem protocol File Transmission System

### KEY

The KEY parameter can be used to encode data being sent or decode data being received. If it is necessary to use a public access bulletin board to send a private file to another user, the sender can encode the file with KEY=N where N is a number from 1 to 255. At some later time, the receiver can download the private file from the bulletin board using the same KEY value used to send the file. The data encryption method is a very simple one and could be broken fairly easily. However, it should deter the casual snoop. The default value for KEY is 0 which produces no encryption.

### LRL

The XMODEM protocol provides no means of passing file attributes (i.e. protection level, logical record length, visible/invisible status, etc.) from the sender to the receiver. The LRL parameter is used in receive mode to set the logical record length of the received file which did not already exist on the receiver's system. This value can be obtained from the sender's directory. The default value for LRL is 0 (256 byte logical record length). The LRL parameter has no effect in send mode.

### NOTIFY

If the NOTIFY parameter is specified, X-FTS will provide an audible indication of the transfer status at the termination of the transfer. If the file transfer was successful, the beep(s) will be short and high pitched. If the file transfer was aborted due to some error, the beep(s) will be of longer duration and lower pitch. This parameter is ideal for long files or for multi-file JCL controlled transfers. The NOTIFY parameter defaults to "OFF" if it is left out of the parameter list. If NOTIFY is specified with no value or if NOTIFY=ON is specified, then NOTIFY defaults to 3 beeps. Any value from 0 to 255 may be specified. NOTE: This parameter is not supported under LDOS 5.1

### QUIET

When X-FTS is running on a "HOST" computer, output sent to the display also goes to the RS-232 device. Since X-FTS normally displays the block number and error count on the screen, this information would be intermixed with the data being sent to the RS-232. Therefore, when X-FTS is being run on a computer in host mode, this parameter must be specified to prevent this intermixing of data. Since X-FTS will be used quite often by a computer running in host mode, it is set-up to allow the user to change the program to force QUIET mode to default to "ON". Simply COPYING FTS/CMD to XFTS/CMD (or RENAMING FTS/CMD to XFTS/CMD) will cause X-FTS to default to QUIET mode. To avoid using the QUIET parameter, use XFTS/CMD in host mode, and FTS/CMD when not in host mode. With either program, you can override the default by explicitly specifying the QUIET option.

## X-FTS - X-Modem protocol File Transmission System

### RETRY

The RETRY parameter allows the user to set the maximum number of attempts that should be made in sending any one block before the transfer is aborted. This parameter also directly affects the amount of "silence" that can occur between blocks before X-FTS declares a "time-out error". The default value for this parameter is 9 which allows up to nine attempts to send a particular block and allows about 25 seconds of delay between blocks. Specifying RETRY=0 will allow an infinite number of block-send attempts and about 6400 seconds of delay between blocks.

### RS-232 PARAMETER SETTINGS

X-FTS uses a binary transfer protocol so any byte value from zero to 255 (decimal) is perfectly valid. Therefore, the RS-232 hardware must be set for 8-bit word size, and no parity. Number of stop bits and baud rate are not critical to the operation of X-FTS.

### JCL EXECUTION OF X-FTS

X-FTS may be used within a JCL file to transfer several files without operator intervention. X-FTS detects that JCL execution is active and modifies its execution somewhat. During JCL execution, nothing will be sent to the screen including the start-up banner. You may over-ride this with the parameter "QUIET=N". A simple JCL procedure could be created using the PARMDIR program from MISOSYS to allow single command multi-file transfers. (See MFTS/JCL on the distribution diskette).

### USING FILTERS ON THE RS-232 DEVICE

Any filtering of the data input from or output to the RS-232 device, will interfere with the proper operation of X-FTS. PRO-X-FTS will detect the presence of any filter(s) and will skip past each filter in the device chain until it finds the actual RS-232 driver. Under LDOS 5.X, there must be no filter on the RS-232 device during the X-FTS transfer.

### OPERATING SYSTEM PECULIARITIES

#### TRSDOS 6.2

Under TRSDOS 6.2, the COMM program uses only the hardware interrupt function provided by the Radio Shack RS-232 interface to receive incoming characters. On entry to X-FTS, the COMM interrupt function is disabled so that no characters are "stolen" by the COMM program. Since PRO-X-FTS uses the Library Overlay Region of memory (x'2600' - x'3000'), it can be executed from within COMM by pressing the three keys <CLEAR>, <SHIFT>, and <0> together. When the transfer is complete, control will be returned to COMM.

## X-FTS - X-Modem protocol File Transmission System

### TRSDOS 6.1 & 6.0

Prior to TRSDOS 6.2, the COMM program used a combination of hardware interrupts and real time clock interrupts to pickup characters from the RS-232. Because of this, X-FTS can not be executed from within COMM. You must exit COMM and execute X-FTS then return to COMM after the file transfer is complete. When receiving files under these versions of TRSDOS, <BREAK> key detection will seem sluggish. Since there is no way to disable KFLAG\$ support (when a 80H is received from the RS-232, a system <BREAK> is simulated) under these systems, <BREAK> key detection is turned off during block receive so that if the file being received contains a byte equal to the current SETCOM (BREAK=value), the transmission won't be terminated. Under all versions of TRSDOS 6, X-FTS may be executed during the execution of a BASIC program with the SYSTEM command:

```
SYSTEM "RUN FTS R filename (parameters)"
```

### DOS PLUS IV

The most common terminal program used under DOS PLUS is the MTERM program sold by Micro Systems Software. Unfortunately, this program uses the RTC interrupt to receive RS-232 characters. X-FTS can not be executed from within MTERM. As with the COMM program on TRSDOS 6.1 & 6.0, you must exit MTERM to execute X-FTS. Under DOS PLUS, X-FTS may be executed from within a running BASIC program with the SYSTEM command, however the DOS library command "RUN" does not exist under DOS PLUS and is not needed:

```
SYSTEM "FTS R filename (parameters)"
```

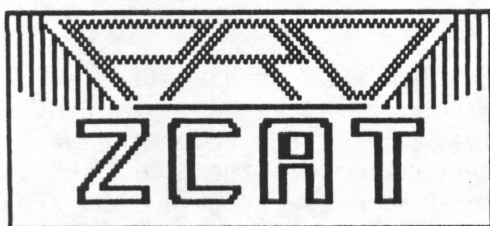
### LDOS 5.1.X

The LCOMM terminal package supplied with LDOS 5 does not have the provision for executing a DOS command while running LCOMM. Therefore, you will have to exit LCOMM to execute X-FTS and then return to LCOMM when the file transfer is complete. Do not specify the BREAK parameter when you invoke the RS232x/DVR driver as this will invariably result in a false <BREAK> detection as described in the TRSDOS 6.0/6.1 discussion above. X-FTS may be executed from LBASIC as follows:

```
CMD"FTS R filename (parameters)"
```



# ZCAT - Disk Catalog Utility



Copyright 1983 by Karl A. Hessinger - MicroConsultants  
Published by MISOSYS, Alexandria, Virginia

## TABLE OF CONTENTS

General Information . . . . .	2
Distribution Diskette . . . . .	2
Invoking ZCAT . . . . .	3
Adding disks to a catalog . . . . .	4
Updating a cataloged disk . . . . .	5
Changing a disk's name . . . . .	5
Removing a cataloged disk . . . . .	6
Searching for a file . . . . .	6
Displaying a cataloged disk directory . . . . .	7
Listing cataloged disks' names . . . . .	8
Printing catalog listings . . . . .	8
Saving changes and exiting to DOS . . . . .	9
Error Message Explanations . . . . .	10

Note: LDOS is a trademark of Logical Systems Incorporated  
TRSDOS is a trademark of Tandy Corp.

## ZCAT - Disk Catalog Utility

### GENERAL INFORMATION

=====

We know you have built up a collection of hundreds of diskettes and need an easy way to locate that particular program and file. We know you want fast response and speed! So rather than waste time trying to determine what DOS each of your disks is compatible with, we recognize that the LDOS user stays with LDOS. ZCAT/PRO-ZCAT is a very fast machine language program that creates and maintains a catalog of all files which reside on your LDOS (or TRSDOS 6.x) formatted diskettes. Each catalog file can store the directory file information on up to 255 diskettes or disks. Approximately 2000 file specifications are supported per catalog file. This number varies with the amount of memory available.

This package is all you need to get a handle on your disk collection via the catalog file or files. It is menu driven for ease of use. The "GETSPEC" initial menu allows you to specify which drive contains your catalog files [a catalog file uses an extension of "/CAT" and is the data base for your disk directories]. Once you specify the drive number, ZCAT displays all files on that drive with a "/CAT" extension. You then enter the name of the catalog file you wish to read or create. Once the specified catalog file has been read or created, the master menu will be displayed.

ZCAT is the professional way to search for the diskette containing a desired file. Because the diskette name (pack ID) is so important to isolate specific diskettes, it is important for you to maintain distinct names on your disks when formatting them. ZCAT or your DOS "ATTRIB" command can be used to rename a diskette where that function becomes necessary.

### DISTRIBUTION DISKETTE

=====

This documentation covers the operation of both the Model I/III LDOS version (ZCAT) and the LDOS 6.x or TRSDOS 6.x compatible version (PRO-ZCAT). The ZCAT package is provided on a 35-track single density data diskette for LDOS Version 5.1. The PRO-ZCAT package is provided on a 40-track single density data diskette for LDOS/TRSDOS Version 6.

## ZCAT - Disk Catalog Utility

### INVOKING ZCAT

=====

The ZCAT utility allows you to create and maintain a catalog of all files which reside on any LDOS or TRSDOS 6.x compatible formatted diskette. It is invoked via the syntax:

ZCAT (INV,SYS,PAGE=nn)

INV Allows the cataloging of invisible files.  
Defaults to OFF.

SYS Allows the cataloging of system files.  
Defaults to OFF.

PAGE=nn Sets the printed page length (default=66)

Abbreviations: I=Inv, S=Sys, P=Page

ZCAT/CMD is a machine language program which will allow the rapid creation and maintenance of a catalog of files. When ZCAT is typed from DOS Ready, the program will load and display the initial logo and version number.

After the initialization has been completed, you will be prompted with the "GETSPEC Menu" as follows:

Enter drive ( 0 - 7 ) or <BREAK> to exit .

Selecting a number <0 - 7> will display all files on that drive with a /CAT extension. "CAT" is the default extension of directory catalog files for use with ZCAT. Pressing <ENTER> will bypass this prompt. Pressing <BREAK> will return you to DOS Ready. The drive number which you enter will also be used as a default drivespec for the "Catalog filespec" response.

You will then be prompted to enter the name of a catalog file with:

Catalog filespec ? .....

Enter the name of the catalog file you wish to read or create. The file specification may be composed of a file name of up to eight characters in length with an optional drive specification. The extension of "/CAT" will be added to the file specification automatically by ZCAT.

After the file specification has been entered, the master menu will be displayed and ZCAT will read in the catalog file if it already exists. Because the maximum number of files which ZCAT can hold changes with the amount of free memory available, it is possible to get the error message:

\* \* File too large for available memory \* \*

If this occurs, press <ENTER> to return to the master menu and <E>xit to DOS. Reduce the amount of high memory which is allocated to DOS and re-run ZCAT.



## ZCAT - Disk Catalog Utility

Once the catalog file has been read, the MASTER MENU will be displayed.

```
*> M A S T E R   M E N U <*
```

```
<A>dd disk to list  
<U>pdate disk in list  
<C>hange a disk's name  
<R>emove a disk from list  
<S>earch for a file  
<D>isplay a disk's files  
<L>ist disks on file  
<P>rint files in list  
<E>xit to GETSPEC
```

Selection ?

CAT file : MARC/CAT:0	Files cataloged : 324
Disks cataloged : 15	Maximum # files : 2226

The desired function may be selected by pressing the letter of the function which is bracketed between the "<>" symbols. In this display, the "DIR file" field will display the current catalog file specification [MARC/CAT is shown for illustration], the "Disks cataloged" field will display the quantity of disks cataloged in the catalog file, the "Files cataloged" field will contain the total number of file specifications cataloged, while the "Maximum # files" field will display the upper limit based on the memory currently available. The following sections will explain the use of each of the functions.

### <A>dd disk to list

=====

The <A>dd function will scan a disk that has not been previously cataloged and add the disk to the catalog list. All non-system visible files that reside on the disk will be cataloged [invisible and or system files may be cataloged if those options are selected at the invocation of ZCAT]. Press the <A> key from the master menu and you will be prompted:

Scan which drive ( 0 - 7 ) ?

Enter the number of the drive which contains the disk you wish to be scanned, or press <ENTER> to scan the last accessed drive. The identification of the last accessed disk drive will be displayed at the bottom of the screen. After the drive has been selected, ZCAT will scan the directory and add all of the non-system, visible files to the list [see ZCAT's INV and SYS command line parameters]. If the disk has already been cataloged the error message:

\* \* Disk is ALREADY cataloged \* \*

will be displayed. You may press <ENTER> to return to the master menu. Remember, each disk cataloged must have a name that is unique to the disk.

## ZCAT - Disk Catalog Utility

After the disk has been cataloged, ZCAT will sort the directory list and will again prompt you for the drive to scan. This gives you an easy way to catalog a number of diskettes via one keystroke. To return to the master menu, press <BREAK> at the prompt. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the "<E>xit to GETSPEC" command, under your control. This is a safeguard for your protection.

### <U>pdate disk in list =====

The <U>pdate function will scan the directory of an ALREADY cataloged disk. It will update the directory file to reflect any changes in the free space on the disk or any changes in the contents of the disk. Press the <U> key from the master menu and you will be prompted:

Scan which drive ( 0 - 7 ) ?

Enter the number of the drive which contains the disk you wish to be scanned, or press <ENTER> to scan the last accessed drive. The drive last accessed will be displayed at the bottom of the screen. After the drive has been selected, ZCAT will scan the directory and update the directory list to reflect any changes that have been made since the disk was last <U>pdated or <A>dded. If the disk has NOT already been cataloged, the error message:

\* \* Disk is NOT cataloged \* \*

will be displayed. You may then press <ENTER> to return to the master menu. Remember that a disk must be <A>dded before it can be <U>pdated.

After the disk has been scanned, ZCAT will sort the directory list [in case files have been added to or deleted from the diskette] and will again prompt you for the drive to scan. This provides you with an easy method to update the catalog for a quantity of diskettes via one keystroke. To return to the master menu press <BREAK> at the prompt. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the "<E>xit to GETSPEC" command, under your control. This is a safeguard for your protection.

### <C>hange a disk's name =====

The <C>hange function will allow you to change a diskette's name from within ZCAT. Press <C> from the master menu and ZCAT will prompt:

Which drive contains disk ( 0 - 7 ) ?

Pressing <BREAK> will return you to the master menu. After the drive has been selected, the current disk name will be displayed. You will be prompted for the NEW disk name. Enter the new disk name or depress <BREAK> to return to the master menu without changing the disk name. After the name has been changed, ZCAT will wait for <ENTER> to be depressed before returning to the master menu.

## ZCAT - Disk Catalog Utility

### <R>remove a disk from list

=====

The <R>remove function will delete all traces of the disk from the catalog's directory list. Press <R> from the master menu and ZCAT will display the names of all disks which are currently in the directory list. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue the display or press <BREAK> to be prompted for "Disk name ?". Enter the name of the disk you wish to remove from the list or press <BREAK> to return to the master menu. If the disk name cannot be found you will be prompted:

\* \* Disk name NOT found \* \*

Press <ENTER> to return to the master menu.

After the diskette has been removed from the catalog's directory list, the list will be sorted and you will be prompted to press <ENTER> to return to the master menu. This function only changes the working copy of the catalog in memory. The actual changes to the catalog file are made via the "<E>xit to GETSPEC" command, under your control. This is a safeguard for your protection.

### <S>earch for a file

=====

The <S>earch function will allow you to rapidly locate a file or a group of files in the directory list. Pressing <S> from the master menu will prompt the question:

Search string ? .....

Enter the search string or press <BREAK> to return to the master menu. The search string may be a filename, a partial filename, or an extension. The search string may also contain a wild card character ("\$\_") which may be used to mark a position as "don't care".

The partial filespec will display all files that begin with those characters. For example, a search string of "LB" would return any filenames which have the first two characters of "LB".

The extension will display all files which have the same extension. A search string of "/CMD" would display any files with an extension of "/CMD".

The dollar sign ("\$\_") wild-card character, may be used to mark a character position as "don't care". For example, a search string of "F\$\_R" would display all files in which the first character is an "F" and the third character is an "R". Note that the second position can be any character. A search string consisting of a single "\$" will list the entire catalog.

The following are some examples of possible search strings and possible matches:



## ZCAT - Disk Catalog Utility

Search string =====	Possible matches =====
L/CMD	LBASIC/CMD, LPT/CMD
\$A/CMD	BACKUP/CMD, BASIC/CMD
/\$\$T	MOD3/DCT, TEST/TXT

ZCAT will then display all files in the directory list which match the search string. The filename will be followed by an asterisk ("\*") if the file is a Partitioned Data Set. Along with the filename will be displayed the modification date of the file and the disk on which the file is located. ZCAT will pause after displaying a screen full of files. Press <ENTER> to continue the display or press <BREAK> to return to the master menu.

### <D>isplay files on a disk =====

The <D>isplay-files-on-a-disk function will display a list of all files which reside on a particular disk. Press <D> from the master menu and all of the disk names cataloged will be displayed. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue displaying disk names or press <BREAK> to be prompted for "Disk name?". At the disk name prompt enter the name of the disk whose files you wish to view, or press <BREAK> to return to the master menu. To display all files, use the <S>earch function with "\$". If the disk name cannot be located you will be prompted with:

\* \* Disk name NOT found \* \*

You may press <ENTER> to return to the master menu.

If the disk name has been found, ZCAT will display the disk name and the free space available on the disk at the top of the screen. ZCAT will then display an alphabetical list of all files on the disk. ZCAT will pause after displaying a screen full of files. Press <ENTER> to continue the display, or press <BREAK> to return to the master menu. The following information will be displayed for each file:

Filename	( Followed by an "*" if the file is a Partitioned Data Set )
Protection level	( i.e. full, read, exec, etc. )
Logical Record Length	( 1 to 256 )
# of Records	( number of logical records )
Size	( the amount of space that the file takes up on the disk, rounded to the nearest K (1K = 1024 bytes) )
Modification date	( the date the file was last written to )
Disk name	( the disk name where the file is located )

## ZCAT - Disk Catalog Utility

A sample of this listing follows:

```

Disk name : MARC0037
Filespec  Prot  LRL  #Recs  Size  Mod Date  Disk name
=====
ATOD/ASM      Full  256    2    1K  27-Sep-82  MARC0037
CASSCO/ASM    Full  256   34    9K  18-Jun-82  MARC0037
CASSCO/CMD    Full  256    4    1K  18-Jun-82  MARC0037
CC2/CCC       Full  256   46   12K  22-Sep-82  MARC0037
CC3/CCC       Full  256   27    7K  22-Sep-82  MARC0037
CC4/CCC       Full  256   32    8K  21-Sep-82  MARC0037
CC6/CCC       Full  256   18    5K  31-Aug-82  MARC0037
CHGDATE/BAS   Full  256    4    1K  20-Oct-81  MARC0037
DABS/ASM      Full  256    2    1K  27-Sep-82  MARC0037
DADD/ASM      Full  256    3    1K  27-Sep-82  MARC0037
[ listing continues]

```

<L>ist disks on file

=====

The <L>ist function will display all the disks which are currently in the directory list. Press <L> from the master menu and the directory filename will be displayed at the top of the screen. The filename will be followed by a list of all the disk names on file with the free space available on that disk. ZCAT will pause after displaying a screen full of disk names. Press <ENTER> to continue the display or press <BREAK> to return to the master menu. The following illustrates such a listing:

These disks are in catalog file : MARC/CAT:0

Disk	Free	Disk	Free	Disk	Free	Disk	Free
=====	=====	=====	=====	=====	=====	=====	=====
MARC0025	3K	MARC0026	3K	MARC0027	OK	MARC0028	OK
MARC0029	OK	MARC0030	33K	MARC0031	2K	MARC0032	15K
MARC0033	3K	MARC0034	9K	MARC0035	11K	MARC0036	32K
MARC0037	OK	MARC0038	84K	MARC0039	54K		

<P>rint files in list

=====

The <P>rint function will allow you to produce a hardcopy of your directory list. Press <P> from the master menu to obtain the print menu.

\*> P R I N T <\*

<F>iles by disk order  
 <E>xpanded file order  
 <C>ompressed file order

Selection ?

## ZCAT - Disk Catalog Utility

Select the type of printout you would like by pressing the first character of the name or press <BREAK> to return to the master menu. You will be prompted:

Press <ENTER> when paper is set to top of form

When the paper has been positioned such that printing will begin on the first line of the paper, press <ENTER> to begin printing. Printing may be aborted at any time by pressing <BREAK>.

<F>iles by disk order  
-----

The disk names will be printed in alphabetical order. Printed with each disk name will be the amount of free space currently available on each disk and an alphabetical list of all files on that disk.

<E>xpanded file order  
-----

Files will be printed alphabetically, one across and 50 per page. The information printed for each file is the same as the "Display Directory" screen listing.

<C>ompressed file order  
-----

Files will be printed in alphabetical order, two per line and 100 per page. The following information will be printed for each file:

Filename	( the name of the file )
Modification date	( the date the file was last written to )
Disk name	( the disk name where the file is located )

<E>xit to GETSPEC  
=====

Press <E> to return to the GETSPEC menu. If changes have been made to the catalog file list, you will be prompted:

Save changes ?

Press <Y> to save the changes or press <N> to return to GETSPEC without saving the changes. At the GETSPEC menu you may press <BREAK> to return to DOS Ready or you may select another catalog file.



## ZCAT - Disk Catalog Utility

### ERROR MESSAGES

=====

#### \* \* Disk is ALREADY cataloged \* \*

This error will occur when the disk name of disk which is being <A>dded is already in the catalog directory list. Use the <U>pdate function if the disk has already been cataloged. **Warning:** this error will also occur if two disks have the same disk name. If such is the case, change the disk name using the <C>hange function and then <A>dd it.

#### \* \* Disk is NOT cataloged \* \*

This error will occur when the disk name of a disk which is being <U>pdated is not in the catalog directory list. Use the <A>dd function to add a disk to the catalog list.

#### \* \* Disk name NOT found \* \*

This error will occur anytime ZCAT cannot locate a disk name in the catalog directory list. Check your spelling of the disk name.

#### \* \* Maximum file limit reached \* \*

This error will occur when the number of files in the catalog directory list becomes equal to the maximum number of files ZCAT can currently hold.

#### \* \* Maximum disk limit reached \* \*

This error will occur when 255 disks have been cataloged in one catalog directory list.

#### \* \* File too large for available memory \* \*

This error will occur when the requested catalog file contains more entries than the current free memory will allow ZCAT to hold. Return to DOS Ready and free up some high memory [you may have to reboot and alter your high memory configuration].

### DOS error messages

If an error occurs during disk I/O the standard DOS error message will be displayed. Press <ENTER> to return to the master menu.

# ZGRAPH

## TABLE OF CONTENTS

GENERAL . . . . .	1
ZGRAPH . . . . .	2
BINCONV . . . . .	14
xxBINCAT . . . . .	16
BINPLAY . . . . .	17
DOSAVE . . . . .	18
BINPRINT . . . . .	19
ZGRAPH FILE FORMATS and EXAMPLES . . . . .	20

Authored and copyrighted (C) 1982/1983 by Karl A. Hessinger.  
ZGRAPH is published by MISOSYS, Alexandria, VA.

LDOS is a trademark of Logical Systems, Inc.  
TRS-80 and TRSDOS are trademarks of Tandy Corp.

### GENERAL

=====

ZGRAPH is a powerful graphics editor package that gives you the tools to construct screen images using your computer's block graphics capabilities. These images may be saved to disk and converted into forms usable by BASIC and machine language programs. Besides the ZGRAPH editor, the package includes five utility programs you can use to create, display, and manage ZGRAPH screens. ZGRAPH does it all: rapidly, totally, and economically!

The ZGRAPH package is provided on a 35-track single density data diskette for LDOS Version 5.1. The PRO-ZGRAPH package is provided on a 40-track single density data diskette for LDOS/TRSDOS Version 6.

## Z G R A P H

### Z G R A P H

=====

The ZGRAPH graphic utility package permits utilization of all of the TRS-80's capabilities in the creation of graphics screens. It will function with those computers supporting TRS-80 block graphics. ZGRAPH is invoked with:

=====	
ZGRAPH	Invoke the GRAPHIC editor
ZGRAPH *	Re-enter ZGRAPH and
<F><A><Y>	abort screen clearing
=====	

### ZGRAPH Editor

=====

The ZGRAPH/CMD file comprises the graphics editor that allows creation of graphic images. When "ZGRAPH" is typed from DOS Ready, the machine language program will load and take control. A graphic logo and a copyright message are displayed during initialization to inform you that ZGRAPH is loading. If you inadvertently exit from ZGRAPH without saving your screen images to disk, you may recover your images by re-entering via "ZGRAPH \*" then issuing the commands, <F><A><Y> to recover the screens.

When ZGRAPH is ready for use, the screen will clear and a flashing graphic cursor will be displayed in the upper left corner of the screen. This is the primary cursor and indicates that ZGRAPH is in the graphics mode. This mode is also the command mode. ZGRAPH possesses two sets of commands, primary and secondary. Primary commands are available anytime the flashing graphic cursor is displayed. Secondary functions are invoked by first depressing <F> and then the appropriate function code. A 'help' list of commands at both levels is available by typing <H> for primary commands or <F><H> for secondary functions. The respective keystrokes will display the complete list of primary or secondary commands available. The <BREAK> key can be pressed to abort most commands.

### Primary Command List

-----

<C>ursor home	<H>elp	<R>everse
<D>raw mode	<I>nsert text	<S>et marker
<E>rase mode	<L>ocate marker	<X>-flip
<F>unction	<M>ove mode	<Y>-flip
	<P>osition	



## Z G R A P H

### Secondary Function List

-----

<A>bort	<F>ill	<M>erge	<R>ectangle	<W>indow
<B>lank	<G>et	<O>utput	<S>ave	<X>change
<C>ircle	<H>elp	<Q>uery	<T>ranslate	<Z>ero
<D>uplicate	<I>nput		<U>sage	<+> Magnify
<E>xit	<L>ine		<V>iew	<-> Reduce

Many ZGRAPH commands and functions (such as <H>elp) overlay the lower portion of the video display screen. Rest assured that your images are not affected. At the point of execution of the command or function, the lower portion of the screen will be restored. The screen may be restored after the <H>elp command by depressing <ENTER>.

### The TRS-80 Graphics Screen

-----

The video display screen of the Model I or Model III TRS-80 consists of 1024 bytes of memory arranged as 16 rows of 64 columns. A Model 4 TRS-80 compatible machine has a display screen of 1920 character cells arranged as 24 rows of 80 columns. Each memory location is capable of displaying one ASCII or special character or any combination of the six (2 wide by 3 high) graphic dots [Note: A Model 4 displays the two lower graphic blocks as 2 wide by 1 high]. These graphic 'dots' will be referred to as pixels (picture elements) in these instructions. When considering the screen as "m" rows of "n" columns of text, the rows are numbered <0 to m-1> starting with the top row and the columns are numbered <0 to n-1> from left to right. In the graphics mode, the pixels are numbered 0 to 2n-1 [127/159], from left to right along the X-axis and 0 to 3m-1 [47/71], from top to bottom. ZGRAPH allows any of the 160 (224 on the Model III/4) possible characters (ASCII, graphic and special) to be displayed at any point on the screen.

### PRIMARY COMMANDS

=====

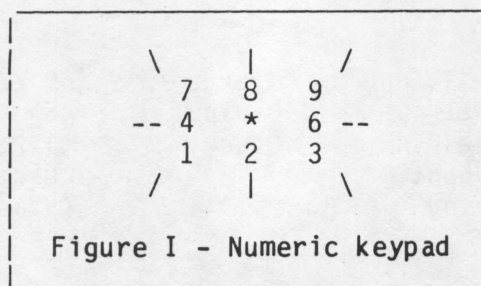
The following paragraphs describe the operation of the primary commands.

#### Cursor Movement

-----

Cursor movement depends on the mode that ZGRAPH is in. In the graphics mode (when ZGRAPH is first entered), movement is achieved using the number keys 1-4 and 6-9 or the four ARROW keys. This allows convenient movement of the cursor using the numeric keypad as shown in Figure I (the regular number keys will also work). The keypad arrangement of:

## ZGRAPH



is directly related to the movement of the cursor. Pressing <4> will move the cursor to the left and <8> will move it up. The corner keys <7>, <9>, <1> and <3> will move the cursor diagonally. All keys will auto-repeat after a short delay. The repeat rate may be set to fast or normal. On the Model I/III, the faster repeat is activated by simultaneously depressing the <SHIFT> key along with the movement key. Under DOS Version 6, the fast or normal rate is set by using the <F1> key to invoke the fast rate and the <F2> key for the normal rate. [On the Model I/III, the keys are also additive; depressing <6> and <9> simultaneously will move the cursor in a direction between right and diagonally to the upper right. Pressing <7>, <4> and <1> simultaneously will move the cursor to the left more rapidly than <4> alone, but pixels will be skipped.] The screen wraps around on all edges. If you go off the screen to the left, you will reappear on the right. The same is true of the top and bottom. The <C>ursor Home command will return the cursor to the upper left corner of the screen. Now that you know how to move the graphic cursor let's examine the three modes that the cursor may be in.

### <D>raw Mode

-----

In this mode, the cursor will leave a trail of bright graphic pixels everywhere it goes. [If you are using the additive feature of the cursor movement keys, the effect will be to produce a dotted line.]

### <E>rase Mode

-----

This mode is the reverse of <D>raw. Everywhere the cursor is moved, the graphic pixels will be turned off (dark). Be sure to cancel this mode by selecting <D>raw or <M>ove as soon as you no longer need it to avoid accidentally erasing pixels.

### <M>ove Mode

-----

This is a non-destructive means of moving the cursor. When in this mode, the cursor may be moved through existing graphic or text locations without disturbing the contents.

## Z G R A P H

### <S>et Marker

-----

Certain functions such as line, rectangle and duplicate require two points of reference to accomplish their job. One reference point is always the current cursor location. The other point is established with the <S>et command. When <S> is depressed, an invisible marker is placed at the current cursor location. The cursor may then be moved to establish the second point of reference. Only one marker may be <S>et at any one time. If you <S>et a new point, the old one will be lost. To view the location of the marker use the <L>ocate or <P>osition commands.

### <L>ocate Marker

-----

This command will cause the <S>et marker to flash for a short interval. When the marker stops flashing, other commands or cursor movement can be executed.

### <P>osition

-----

This command will display the row, column and x,y pixel positions of the cursor and marker. It also displays the current mode. <ENTER> will terminate the display of the positions. Information similar to the following will be displayed when <P> is depressed.

-- Mode --		X - coor	Y - coor	Column	Row
=====		=====	=====	=====	=====
Move	Cursor :	87	48	43	16
	Marker :	29	22	14	7

### <I>nsert Text

-----

This mode is distinguished by an underline cursor. The mode will remain active until <ENTER> is depressed. While in the text <I>nsert mode, cursor movement is via the arrow keys. The cursor is non-destructive of both graphics and text. Simply move the cursor to the desired position and start typing text. The cursor will advance after each letter is entered. The screen wrap-around in the text mode is identical to the graphics mode when the arrow keys are used; however, when text is entered, the line will be advanced if you go off the right edge. The cursor will not move past the bottom edge of the screen. Remember when you see the effect of entering text on the surrounding text, that each text character occupies the same space as six graphic pixels (2 x 3). Three special entries are available in the text mode:

<CLEAR><SPACE> - will enter a graphic blank instead of the normal ASCII blank entered with the space bar. This is important if you <R>everse.

<CLEAR><@> - will enter a full graphic block (chr\$(191)); all pixels on).



## Z G R A P H

<CLEAR><-> - on the Model III/4 only, will display a reverse video question mark. Two hexadecimal digits may then be typed (they will not be displayed) and will be interpreted as the character they represent. This allows access to the Model III special character set. For example, depress <CLEAR><-> then <E><F> and the copyright symbol will be displayed. You will always get the special characters and not the alternate space compression characters.

### <R>everse

-----

This command will reverse the screen video. All bright graphic areas will become dark and vice versa. Text will not (and can not) be reversed. If you want spaces between text to remain dark when the screen is <R>everse, use the space bar when entering text otherwise use <CLEAR><SPACE> as explained under <I> above.

### <X>-Flip

-----

This command will create a mirror image of the screen about the Y-axis. The graphics will be a true mirror image and the order of text characters will be reversed but, of course, the individual text characters cannot be reversed. A second <X>-Flip will restore the screen to its original configuration.

### <Y>-Flip

-----

The same as <X>-Flip as described above except about the X-axis.

## SECONDARY FUNCTIONS

=====

The secondary functions of ZGRAPH are obtained by depressing <F>. At the prompt,

### Function?

the following functions are available (if you decide not to enter a function, depress <ENTER> to return to the command mode).

# Z G R A P H

## ZGRAPH Data Transfer Functions

ZGRAPH has a number of in-memory screen buffers in addition to the video display screen. The number of buffers varies with the amount of memory available. All but one of these buffers are general purpose buffers and are available to the user to store displays. This is useful when creating a large graphic consisting of several ZGRAPH images or in creating those images using the <M>erge function. ZGRAPH can also load and save images to disk files. All data moving to and from the disk passes through the primary video display. The last internal display buffer is best described as the error recovery or auxilliary buffer. When any function is executed that destroys an existing screen display (<B>lank, <G>et, <I>nsert text, <L>oad, <M>erge, etc.) the current display is automatically saved to the auxilliary buffer prior to the function being executed. If you discover that you made an error (<M>erged the wrong display for example), you may recover the original display using the <A>bort command. Figure II below shows how data is moved within ZGRAPH.

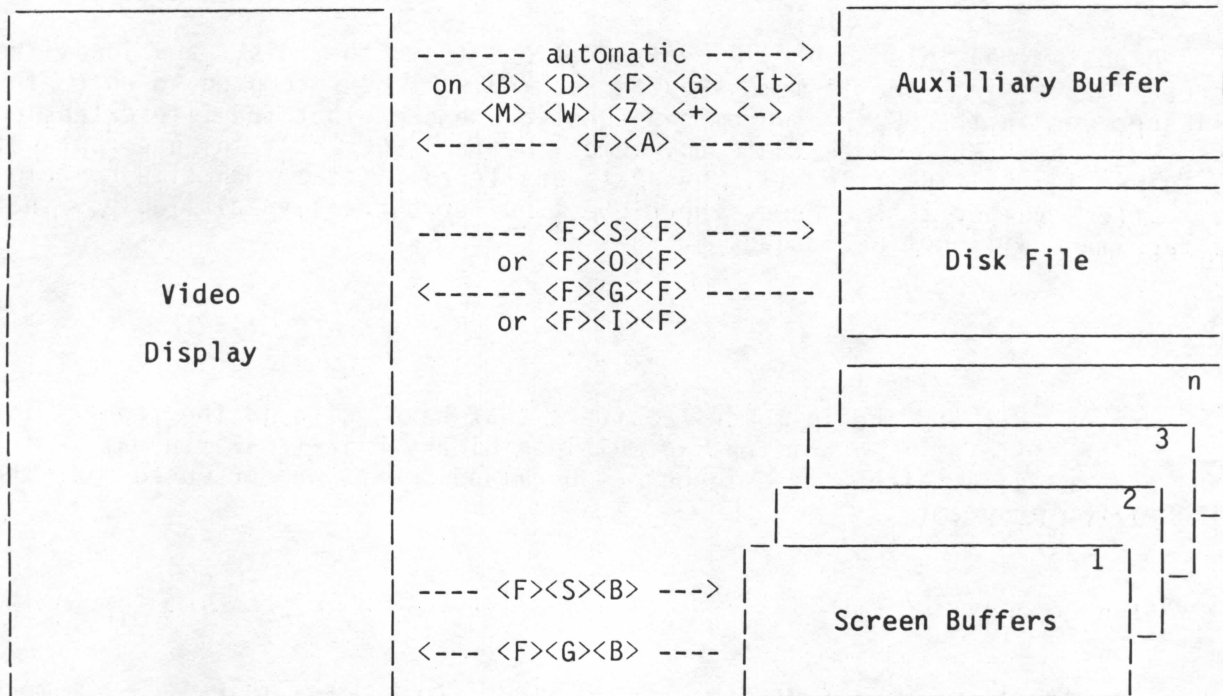


Figure II - Data movement within ZGRAPH

<G>et

Get is the function for loading the video display screen. Pressing <G>et will prompt the question,

Get from <B>uffer or <F>ile?

If <B>uffer is requested, you will be prompted to select a buffer with:

## Z G R A P H

### Buffer number (1-n)?

The current contents of the display buffer will be replaced by the contents of the buffer that you specify. If you request the <F>ile form of get, you will be prompted to enter a filespec via the message:

### Filespec w/o ext for get > .....

Up to ten characters may be entered; eight for the name and two for the drivespec (i.e., testpict:2). An extension of "/BIN" (binary) is assumed for all ZGRAPH binary files. If a drivespec is not entered, all drives will be searched. If no filespec is entered, ZGRAPH will use the last filespec entered with either the <G>et or <S>ave commands. If you accidentally destroy a display by <G>etting another image, the original display may be recovered with <A>bort. The <G>et command may be cancelled prior to execution by depressing <BREAK>.

### <I>nput

-----

Reads a Multiple Binary File (/MBF type) from the disk and loads the screen images into the in-memory buffers. You will be prompted to enter the filespec as in the <G><F> subcommand functions except that the file extension used will be "/MBF". Note that any memory buffers currently in use but not contained in the /MBF file will be left unaltered. If the /MBF file contains any buffer number that exceeds the highest buffer currently available, that buffer image will not be loaded.

### <O>utput

-----

Writes all of the in-memory buffers that have data in them to a disk file. The file will be written in Multiple Binary File (/MBF) format which can be reloaded with the <I>nput subcommand function or used in the BINPLAY/CMD program.

### <S>ave

-----

This function will save the screen image to a disk file or a memory buffer. The disk save will use a rectangular area of the screen described by the cursor home position and the current cursor position. Saving to a buffer will use the entire screen. To save a screen image, first position the cursor to the lower right-hand corner of the area that you wish to save. When <S>ave is depressed at the "Function?" prompt, you will be queried:

### Save to <B>uffer or <F>ile?

The <B> and <F> options are the same as under <G>et. The next prompts will indicate the size of the screen that ZGRAPH is set to save. If you forgot to position the cursor to the lower right corner of the area that you wish to save, answering 'N' to these prompts will return you to the command mode without saving the image. Like <G>et, if no filespec is entered, ZGRAPH will use the last filespec entered with either the <G>et or <S>ave commands.



## Z G R A P H

<S>aving does not affect the contents of the screen.

### <M>erge

-----  
<M>erge allows you to superimpose one image over another. You will be prompted for:

Merge from <B>uffer or <F>ile?

The responses and remaining queries are the same as <G>et. Unlike <G>et, however, the current display is not cleared but rather, the new display is superimposed on top of it. In the merge process, ZGRAPH uses the following rules to establish the merging of a "source" buffer/file to the video display:

1. Any source byte will replace a video graphic blank (X'80').
2. If both video and source bytes are graphic, the source graphic byte will be logically ORed (pixel by pixel) with the video byte.
3. If the video byte is graphic and the source byte is non-graphic, the video will retain its current pixel configuration.
4. If the video byte is non-graphic, the video will retain its non-graphic value.

### <X>change

-----  
This function completes the data manipulation capabilities of ZGRAPH. <X>change will prompt with:

<X>change Screen with Buffer (1-n)?

Enter a buffer number from 1 to n. The <X>change function swaps the contents of the display with the specified buffer (the contents of the screen are placed into the buffer while the old contents of the buffer are placed on the screen). The auxilliary buffer retains a copy of the previous screen image.

## &lt;A&gt;bort

-----

The <A>bort command allows you to recover a screen display that was inadvertently destroyed. For instance, if you <G>et a new image without <S>aving the old one, you can recover with <A>bort. Depressing <A> will yield the query:

OK to load auxilliary buffer?

Answering <Y>es will cause the auxilliary buffer to load into the display. The reason for the question is that while <A>bort will allow you to recover from other catastrophic errors there is no recovery from <A>bort; that is, invoking <A>bort will cause a permanent loss of the current screen display. If you want to preserve the current screen, do a <S>ave just prior to issuing the <A>bort function.

## ZGRAPH Graphic Generation Functions

=====

The following functions are used to generate various images.

## &lt;C&gt;ircle

-----

The <C>ircle function will draw a circle or an arc around the current location of the cursor. You will be prompted for "Radius?" which is in units of pixels along the Y-axis. The next prompt is for "Starting arc (0-7)>". This value represents the number of the arc as illustrated in figure III. The last prompt is for "Ending arc (0-7)?". For instance, an arc from zero degrees to ninety degrees is 0-1. To draw a full circle, specify arc 0 to arc 7. A left half-moon would be arc 2-5. If ZGRAPH is in the <E>rase mode, the arc(s) will be reset rather than set.

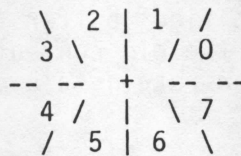


Figure III - Circle function degree diagram

## &lt;D&gt;uplicate

-----

The <D>uplicate command makes use of the marker created with the <S>et command. First <S>et the marker in the upper left corner of the section of the screen that you wish to duplicate. Move the cursor to the lower right hand corner and depress <D>. The block to be duplicated has now been defined. The graphic cursor is replaced with the underline cursor. Position this

## Z G R A P H

cursor (using the arrow keys the same as the insert-text mode with <BREAK> to abort) at the upper left corner of the area where you wish to place the duplicate block and press <ENTER>. As much of the block will be duplicated to the bottom of the screen. If necessary, the duplicated block will wrap around the right edge of the screen. The image contained in the block may be repeatedly duplicated by depressing <D>.

### <F>i11

-----

This function can be used to change pixels enclosed within a boundary. In DRAW and MOVE modes, all pixels will be set until a boundary of set pixels is reached. In ERASE mode, all pixels will be reset until a boundary of reset pixels is reached. If no such boundary exists, the display screen edges will be considered to be the boundary.

### <L>ine

-----

<L>ine will establish the best fitting (straightest) line between the marker <S>et and the current cursor position. In ZGRAPH "move" or "draw" mode, the line will be constructed with set pixels. In "erase" mode, the line will be constructed with reset pixels. The marker position will be updated to the current cursor position after each <L>ine is drawn. The automatic re-setting of the marker will provide an easy way to construct lines connected end-to-end.

### <R>ectangle

-----

To create a rectangle of any size, first <S>et the marker at the upper left corner of the desired rectangle. Move the cursor to the lower right corner and depress <R>ectangle. The four sides of the rectangle will be constructed as "set" or "reset" pixels depending on the mode as in <L>ine.

### <Z>ero

-----

This function is used to fill the rectangle formed by the cursor and the marker with either all pixels ON or all pixels OFF. It is useful for clearing a large block of the screen (or "whiting" a large block).

### <+>Magnify

-----

This function can be used to enlarge a particular rectangular area of the screen display. It will magnify the graphics in the rectangle formed by the marker and the cursor. You will be prompted to enter a magnification ratio in the range <2-5>. For a magnification ratio of "2", each pixel is mapped to a 2x2 pixel group; a magnification ratio of "3" maps each pixel to a 3x3 pixel group; etc.



## &lt;-&gt;Reduce

-----

This function can be used to shrink a particular rectangular area of the screen display. It will reduce the graphics in the rectangle formed by the marker and the cursor from a 2x2 pixel group to a 1x1 pixel group depending on the reduction mode. You will be prompted to enter the reduction mode <1,2>. Mode 1 will set a corresponding pixel if any one of the four pixels in the 2x2 group is set. Mode 2 requires any two of the four pixels to be set before setting the corresponding pixel in the reduction.

## &lt;B&gt;lank

-----

<B>lank gives you the options of clearing the display screen or any buffer. It will display the prompt:

Blank <S>creen, <B>uffer or <A>ll ?

If blank <A>ll is selected, you will be asked, "OK to blank all?" as a double check of your intent. If <S>creen blank is selected, ZGRAPH clears the video display screen. The screen is saved to the auxilliary buffer and may be recovered if the blanking was inadvertant. Note that the blanking function fills the target screen/buffer with graphic blanks (X'80'). If you select the <B>uffer blanking, you will be prompted to select a specific buffer.

## &lt;T&gt;ranslate

-----

This function will translate all occurrences of a character to another character. After typing <T>, respond with the decimal value to translate and the decimal value to result after the translation. The "find" and "replace" character values may also be entered as their single key entry in addition to their decimal ASCII value. For example, the letter "A" may be entered as either "65" or "A" (without the quotes). Note that <CLEAR-SPACE> and <CLEAR-@> (graphic block 191) are equally acceptable as "single key entries". You can rapidly change all text blanks to graphic blanks with this command.

## &lt;W&gt;indow

-----

The first thing you will notice when you depress <W>indow is that the cursor disappears. While in the window mode, the entire screen display will move in response to the arrow keys. Any part of the image moved off of the edges of the screen is lost. To terminate the window mode, press <ENTER>. This command is very useful to reposition an entire image on the screen.

## Z G R A P H

### <E>xit

-----

This function provides a graceful exit from ZGRAPH to DOS Ready.

### <U>sage

-----

This function is used to obtain the status of the in-memory buffers. It displays a graphic block next to all buffers which have data in them. The graphic block "flag" is set by <X>change and <S>ave, and is reset by <B>lank.

### <V>iew

-----

This function allows the rapid display of the memory buffers contents to the video display screen. You will be permitted to specify the buffer range, and a relative speed at which the buffers will be displayed, <1-9>. <V>iew may be aborted by depressing <BREAK>.

### <Q>uery

-----

<Q>uery is used to question the operating system. On the Models I/III, it obtains directory information. <Q>uery will issue two prompts as follows:

Directory of which drive ?  
Display /<B>IN or /<M>BF files ?

The first prompt is used to specify the drive number for the directory information. The second is to specify whether you want to see the names of the binary or multiple-binary files. It will then list a directory of all files on the specified drive that possess the specified extension. The drive volume name and available space in "K" will also be displayed.

Under DOS Version 6, the <Q>uery command is used to access any DOS library command. <Q>uery will issue one prompt as follows:

Command? .....

You can then enter any library command such as "DIR /BIN:2". When the DOS command completes, you can return to the ZGRAPH screen by depressing the <ENTER> key.

## B I N C O N V

## BINCONV: ZGRAPH File Conversion Program

The post-processing program, BINCONV/CMD, has been provided to allow ZGRAPH created displays to be used in other applications. BINCONV will translate the ZGRAPH binary file format to other formats. It is invoked by entering the command:

BINCONV	Run conversion program
---------	------------------------

ZGRAPH's standard file format is a pure binary representation of the screen display. Each line of the screen memory is <S>aved as the values of the memory bytes terminated by a carriage return (x'0D'). A screen saved with ZGRAPH would thus occupy a number of bytes equal to the number of rows multiplied by one greater than the number of columns.

The BINCONV program will display the following menu of choices:

```
* * ZGRAPH file conversion utility * *
```

- ```
<1> - ZGRAPH to Load Module
<2> - ZGRAPH to Packed BASIC
<3> - ZGRAPH to BASIC Data
<4> - ZGRAPH to EDAS
<5> - Disk directory
<6> - Exit to DOS
```

Depress the appropriate number key for the format you desire. The formats are as follows:

## ZGRAPH to Load Module

The following prompts must be answered under this format mode:

```

Starting address { 15360/12288 } ?
[For 5.1.3; Transfer address { 73 } ??]
Filespec to convert ( w/o ext ) ?

```

For the Model I/III, the starting address defaults to the start of the video display memory and the transfer address defaults to the system vector, @KEY. The effect of choosing the default values (depressing <ENTER> in response to the prompts) is to create an executable /CMD file that will place your image on the screen and pause until a key is depressed, at which time it will return to DOS. Under DOS Version 6, the file is constructed as a core image which loads at the start of user RAM and returns to DOS Ready.



### ZGRAPH to Packed BASIC

-----

This format mode will prompt for the following:

```

        Array name { ZG } ?
        Starting index { 0 } ?
        Starting line number { 100 } ?
        Line number increment { 10 } ?
        Input filespec ( w/o ext ) ?
        Output filespec ( w/o ext ) ?
    
```

The default values will create a file with the extension of "/BAS". The file will consist of packed graphics strings with each line consisting of the string {ZG\$(#)="packed value of one line of your image"} starting with an index (#) of 0, line number of 100 and line number increment of 10.

### ZGRAPH to BASIC Data

-----

The following prompts must be answered:

```

        Starting line number { 100 } ?
        Line number increment { 10 } ?
        Input filespec ( w/o ext ) ?
        Output filespec ( w/o ext ) ?
    
```

This option will create BASIC data statements starting with line 100 (if the default is used). Each statement will consist of 16 decimal numbers representing the sequential values of your screen image. The file will be saved with an extension of /BAS to allow merging into your BASIC program.

### ZGRAPH to EDAS

-----

Prompts similar to "ZGRAPH to BASIC Data" will appear. The file that will be created will have an extension of /ASM and will be in the EDAS editor/assembler format. Each line of the file will consist of a DEFB statement and 16 decimal values representing the values of the bytes of your image. This file may then be merged into an assembler program. Under LDOS Version 5, the file will be headered and line numbered. Under LDOS/TRSDOS 6, the header and line numbers will be omitted.

### Miscellaneous

-----

The fifth menu option gives you the capability of displaying a disk directory [under LDOS Version 5] or of executing a DOS command [under LDOS/TRSDOS Version 6]. This is identical to the <Q> function of ZGRAPH.

To exit BINCONV, press <6> from the main menu.

# x x B I N C A T

## xxBINCAT/CMD

=====

These programs are used to print a single binary file or a concatenation of single files to a printer. The RSBINCAT program supports the Radio Shack DMP printers. The EPBINCAT program supports the Epson printers. xxBINCAT is invoked with the command:

```
=====
|
|  xxBINCAT (ADDLF,DENSE,RS2100)  Run the printing program.
|
|  ADDLF      - An EPBINCAT parameter used to force a line
|                feed after a carriage return.
|
|  DENSE      - If entered, the entire graphic will be printed
|                in boldface via overstrike.
|
|  RS2100     - A parameter to be entered if the DMP2100
|                printer is being used (RSBINCAT only).
|
|  Abbreviations: A=ADDLF, D=DENSE
|
|=====
```

The BINCAT program will print the graphics cells stored in one or more ZGRAPH binary (/BIN) files. If more than one input file is specified, BINCAT will concatenate the images so they are printed left to right. Therefore, it is possible to combine two or more screen images to make a larger "picture". For instance, a large picture made up of six screens in a three across by two high "picture" can be printed by first concatenating and printing the three top images then concatenating and printing the three lower images.

When BINCAT is first invoked, it will display the prompt:

**ZGRAPH filespec w/o ext ?**

Enter the file specification of a ZGRAPH binary file. BINCAT will continue to prompt for additional file specifications until you depress <ENTER> by itself to end the input. Each file identified will be concatenated for printing across the page. If you depress the <BREAK> key, BINCAT will exit to DOS Ready. Once all file(s) are entered, ZGRAPH will prompt with:

**Enter magnification (1-9) ?**

The magnification value will cause printing to use as many dots vertically and horizontally for printing a pixel as specified by the value. For example, a magnification of "3" will print each screen pixel in a 3-dot by 3-dot impression. When the printing is complete, BINCAT will give you the opportunity to print another copy of the concatenated image. It does this by returning to the "Enter magnification" prompt thus allowing you to specify any desired magnification. If you respond with <ENTER> by itself, BINCAT will return to the "Input filespec" prompt. If you enter a <BREAK>, BINCAT will exit to DOS Ready.

# B I N P L A Y

## BINPLAY/CMD

=====

This program will perform a video display sequence of all buffers saved in a ZGRAPH Multiple Binary File (/MBF). It is invoked via the command:

```
=====
BINPLAY filespec (PAUSE,DELAY,REPEAT) Invoke MBF play
filespec - Specifies the file containing the screen
           images. The default extension is "/MBF".
DELAY=val - "Val" is specified as the relative length
            of time a screen is displayed. The range of
            values acceptable is <0-255>. Default is 128.
PAUSE=sw  - If "sw" is specified as <ON>, then BINPLAY
            will wait for you to depress <ENTER> before
            displaying the next frame. Default is OFF.
REPEAT    - Is specified to invoke a repeating play
            of all screens until <BREAK> is pressed.

Abbreviations: D=DELAY, P=PAUSE, R=REPEAT
=====
```

The BINPLAY program is useful for displaying a small series of graphic screens. It will display the sequence of screens saved from the ZGRAPH in-memory buffers that are in use at the time that the ZGRAPH <O>utput function is specified.

The DELAY parameter allows you to set the "viewing" time of each screen. The PAUSE parameter is used to suppress the automatic advance to each screen. This mode is useful when previewing a Multiple Binary File. If the screens are designed as a repeating "slide" show, then the REPEAT parameter will cause the play to automatically repeat the image sequence. It will play continuously until <BREAK> is pressed.



# DOSAVE Screen Saving Filter

DOSAVE is a keyboard filter that is similar to the DOS screen print function. However, where the screen print directs an image of the screen to the printer, DOSAVE will direct the screen image to a disk file specified by the user. The screen saving filter, DOSAVE, is established by entering the command(s):

|                                                           |                                             |
|-----------------------------------------------------------|---------------------------------------------|
| For LDOS 5.1<br>FILTER *KI using DOSAVE                   | Enable screen saver                         |
| For LDOS 6.x<br>SET *DS to DOSAVE<br>FILTER *KI using *DS | Reside filter module<br>Enable screen saver |

Once established, depressing <CLEAR><SHIFT><S> will activate the filter. The prompt,

Filespec?

will appear on the screen. Enter the desired file specification (a default extension of "/BIN" will be applied). The contents of the screen will be saved in the standard ZGRAPH binary format. These screen files may be loaded into ZGRAPH for further operations.

# B I N P R I N T

## BINPRINT File Printing Program

=====

This program provides the capability of printing a binary graphic file to a printer that supports compatible block graphics (i.e. MX-80 with GRAFTRAX or other MX printers with the ALTCHAR printer driver that is part of the GRASP package). Printing is invoked with the command:

```
=====
|
|  BINPRINT filespec (OFFSET,STRIP=value)
|
|  OFFSET          Used for non-GRAFTRAX MX-80 printers.
|
|  STRIP=value     Will convert any character above "value"
|                  to a blank (X'20').
|
|  Abbreviations: O=OFFSET, S=STRIP
|
=====
```

Typing "BINPRINT filespec" from DOS Ready will cause the specified file to be sent to the printer. The default extension, /BIN, will be used if none is specified. Note that the DOS screen print function <CONTROL><\*> remains active in ZGRAPH and may also be used if this feature has been selected using the SYSTEM(GRAPHIC) command [and the screen print (JKL) option of KI/DVR for the Model I/III LDOS 5.1 user].

The parameter, OFFSET, is used to add the decimal value 32 to graphics codes in order to place the code value into the range proper for those printers supporting the TRS-80 graphics but at a CHAR+32 value.

## ZGRAPH FILE FORMATS

=====

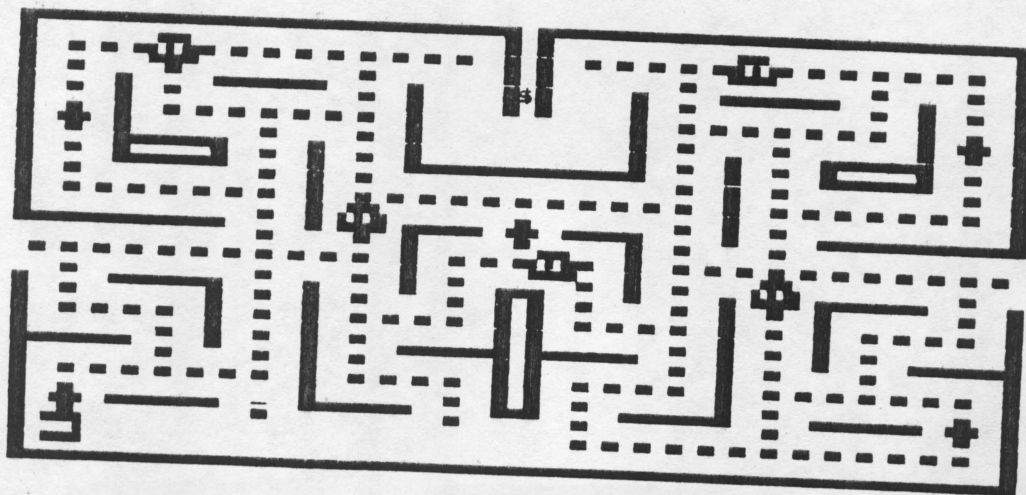
The "/BIN" binary format file is composed of each video row of characters terminated by a carriage return, <ENTER>. It will contain from one to n rows of data based on the location of the cursor when the file was saved.

The "/MBF" multiple binary format file stores a number of memory-buffer images. It uses the first sector as a flag field to indicate which buffers are saved in the file. Relative byte 0 stores the length of the flag field <1-255>. Relative bytes 1-255 will contain an X'01' if the next image in the file stores the corresponding buffer. An X'00' indicates that the buffer is unused and no image exists for it in the /MBF file. Each image is contained in a 1K (Model I/III) or 2K (Version 6) block [i.e. 4 sectors or 8 sectors]. The first image corresponds to the first flag containing an X'01'; the second image corresponds to the second flag containing an X'01'; and so forth.

## EXAMPLES

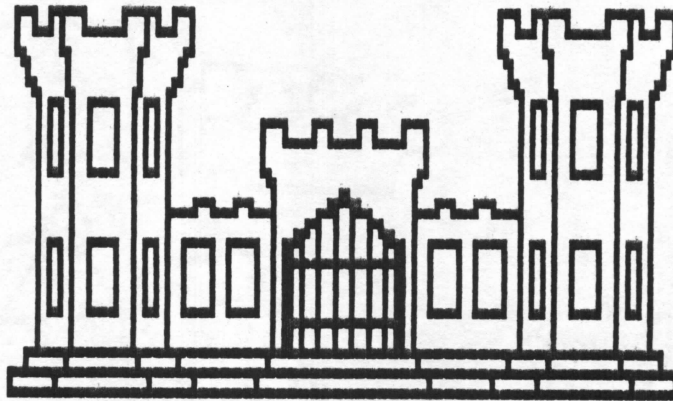
=====

The following examples were generated using ZGRAPH. The printing was accomplished using both Epson MX-80 GRAFTRAX and MX-100 printers. The Graphic Support Package (GRASP) available from MISOSYS was used to implement pixel graphic printing on the MX-100 for both the 10-pica and 12-pica graphics. The /BIN files for these graphics are included on your ZGRAPH diskette.



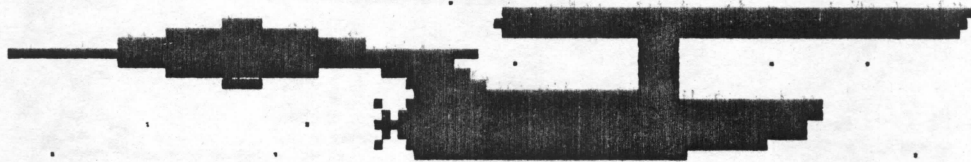
PACMAN: MX-100/ALTCHAR-STD12





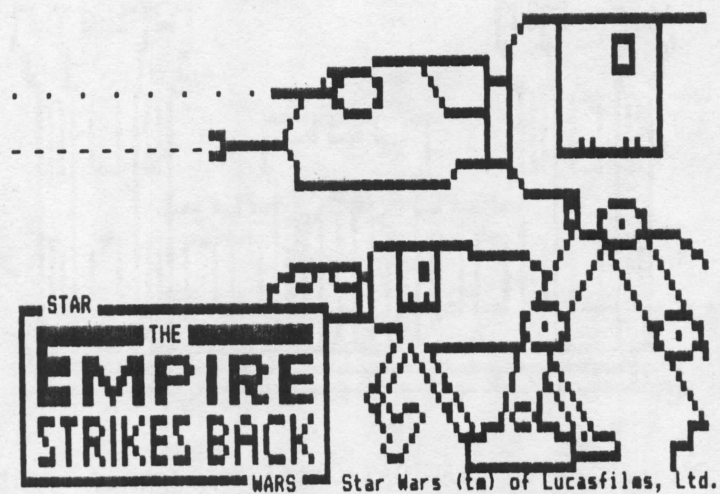
CASTLE: MX-80 GRAFTRAX/Condensed-double strike

STAR TREK

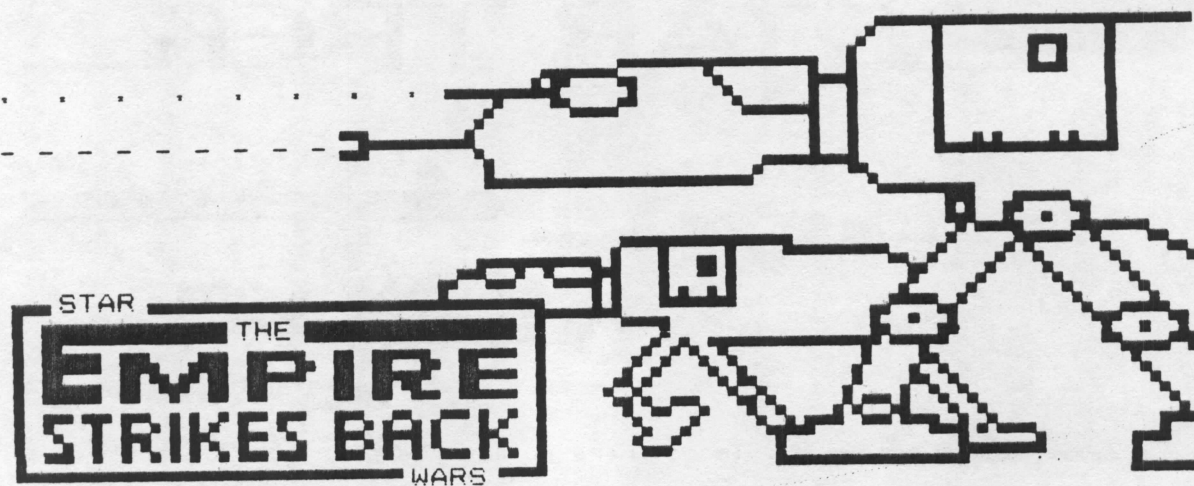


Our Mission..... To go where no man has ever gone before.....@

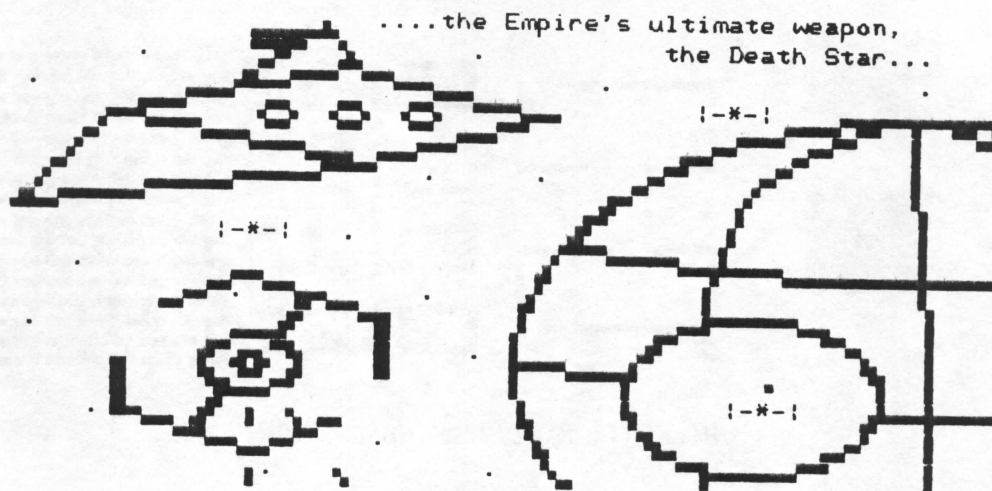
STARTREK: MX-100/ALTCHAR-STD12



EMPIRE: MX-80 GRAFTRAX/Condensed-double strike



EMPIRE: MX-100/ALTCHAR-STD10



DEATHSTR: MX-100/ALTCHAR-STD12

A long time ago in a galaxy far, far away....

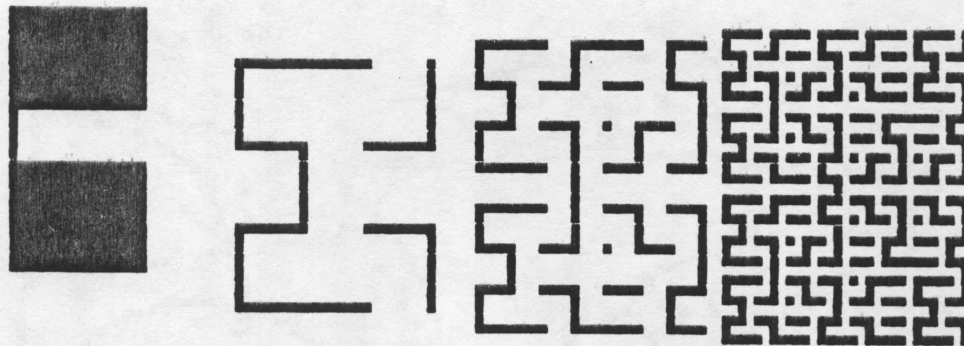
STAR  
WARS

Star Wars (tm) of Lucasfilms, Ltd.

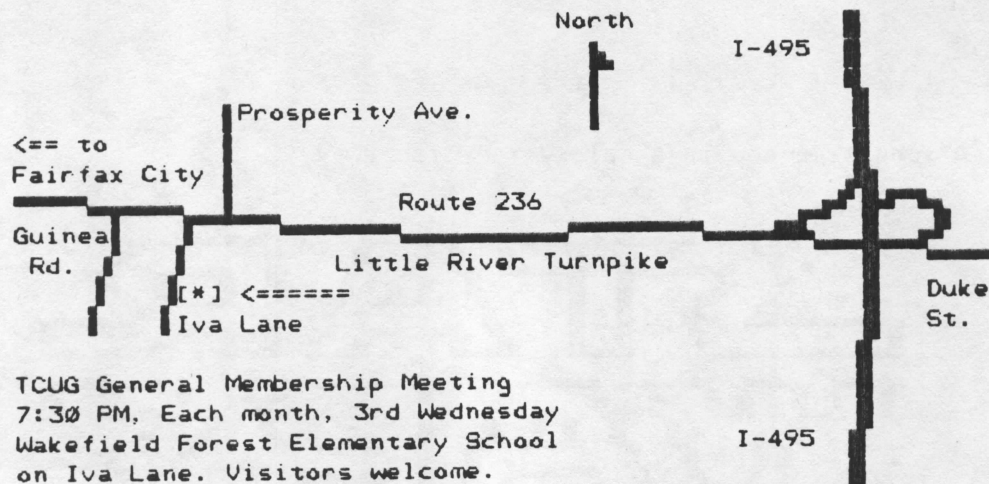
STARWARS: MX-100/ALTCHAR-STD12



HILBERT CURVES



HILBERT: MX-100/ALTCHAR-STD12



TCUGMAP: MX-100/ALTCHAR-STD12

# ZSHELL

Copyright (C) 1983, 1984, 1985 by Karl A. Hessinger, MicroConsultants  
Published by MISOSYS, Sterling, Virginia

Who has not come across a BASIC program that was loaded with "PRINT" statements while that brand new printer was just connected to the machine. Well, time to change all of those "PRINT" statements to "LPRINT". Perhaps you learned the old technique of poking certain values into memory locations said to be the "Video Device Control Block" and were thus able to "modify" your program easily. In any event, you progressed to purchasing DOS and discovered this wonderful ROUTE command. With it, you could "ROUTE \*DO to \*PR" and run that BASIC program with the PRINTs mysteriously changed to LPRINTs. All you had to do was to "RESET \*DO" after your program completed and returned to "DOS Ready".

DOS was a generation ahead of the "poking". There is, however, a more modern technique of dealing with the problem of wanting program output to go somewhere other than where the program was originally intending it to go. Or, for that matter, input to a program. ZSHELL is that next step. We have provided three significant features in this product. ZSHELL and your DOS system take you to another level of convenience and flexibility with program input/output.

## TABLE OF CONTENTS

|                                     |    |
|-------------------------------------|----|
| DISTRIBUTION DISKETTE . . . . .     | 2  |
| WHAT IS ZSHELL ANYWAY . . . . .     | 2  |
| WHAT CAN I DO WITH ZSHELL . . . . . | 3  |
| INSTALLING ZSHELL . . . . .         | 4  |
| COMMAND LINE SYNTAX . . . . .       | 5  |
| ADVANCED TOPICS . . . . .           | 8  |
| ERROR MESSAGES EXPLAINED . . . . .  | 9  |
| KISTORE FILTER . . . . .            | 11 |
| WILDCARD . . . . .                  | 12 |

## DISTRIBUTION DISKETTE

The ZSHELL package is distributed on a 35-track single density LDOS formatted data diskette. This diskette is for use under LDOS 5.1 on either a Model I or Model III. The PRO-ZSHELL package is distributed on a 40-track double density data diskette and is for use under TRSDOS 6.x. The master diskette contains three programs: ZSHELL/CMD, KISTORE/FLT, and WC/CMD. This documentation pertains to the use of ZSHELL under either DOS environment.

## WHAT IS ZSHELL ANYWAY?

Almost every program needs some type of input and produces some type of output. In most cases, the input is retrieved from the keyboard, or in DOS, the \*KI device (the "KI" stands for Keyboard Input while the asterisk, "\*", is prefixed to indicate a device specification). Any output is then displayed on the video, or the \*DO device (Display Output). ZSHELL is a system enhancement which will allow you to temporarily redirect the input or output from or to any file or device instead of the normal \*KI or \*DO devices until the program being executed returns to "DOS Ready".

Every one of us has a program which gets some input from the keyboard and then displays its output on the video. What do you do if you want the output to go to the printer (\*PR)? You'd probably sit down and rewrite the program to change all the PRINT statements into LPRINT statements. Well, that leaves you with two different programs that do almost the same thing, except that their output goes to two different places. Well somehow that just doesn't seem fair to have to have two programs, so you whip out your DOS manual and find the ROUTE command. Aha, I can write one program to do the computing and use the ROUTE command to cause the output to go to the \*DO or the \*PR without having two different programs.

OK, that sounds good to me. But what happens when the program finishes, and everything that should be displayed on the screen is still going to the printer? Well you have run into one of the primary disadvantages to using the ROUTE command. To unroute the device you use the RESET command; however, the RESET command not only will undo the route, it will also remove any drivers or filters from the device. This inability to unlink the ROUTEing but leave the drivers and filters associated with the device unaltered (the device chain), can be overcome by using ZSHELL. ZSHELL can perform the same redirection but can limit it to the duration of the program's execution. After the program's execution has terminated, all of the redirection will be removed and all of the devices will be left as they were before the redirection was done.

The process of causing keyboard input or video output to go to somewhere other than is normally expected (\*KI or \*DO) is called "I/O redirection". For the remainder of these instructions the \*KI will be called the standard input and the \*DO will be called the standard output. With ZSHELL, you can tell your computer where to get the standard input or where to put the standard output or both at the same time.

Now that you have an understanding of I/O redirection, we will advance to the next major capability of ZSHELL, piping.



You don't have to be a professional plumber to know that you have pipes running through your house or apartment. These pipes are used to distribute the water throughout your dwelling. ZSHELL, like your house, uses pipes, but instead of carrying water they carry information. With ZSHELL it is possible to cause the standard output of one program to become the input of another program. The information is said to be "piped" between the programs. Just visualize ZSHELL as an imaginary pipe carrying the information from one program to another.

ZSHELL's last ability is quite different from the first two and much easier to understand. ZSHELL allows you to enter more than one command on a command line - all you have to do is separate the commands with a semi-colon. The first command will be executed. When it has completed, the next command will be executed. The only limit to the number of commands is the length of the command line - which can be up to 255 characters in length with ZSHELL.

#### WHAT CAN I DO WITH ZSHELL?

Well now that you have a basic understanding of I/O redirection and piping just how would you go about using them? To help you to understand what you can do, let's look at some examples.

##### Example # 1

The personal finance program you just completed, writes all of its output onto the video but you would like to have a copy appear on your printer. Instead of rewriting your program you run your program and instruct ZSHELL to redirect your output to the printer instead of the video.

##### Example # 2

You have a program which takes all its input from the keyboard and you would like to use it under JCL. After reading the DOS manual you find that since the program uses INKEY\$ to scan the keyboard, JCL won't function. You can still accomplish what you want by making a file of the necessary keystrokes, run the program and instruct ZSHELL to route all standard input from that file.

##### Example # 3

You are trying to load a document which has tab characters (ASCII 9) into a text editor which doesn't "understand" the tab characters. You then remember that the LIST command has an option to expand tabs. Instruct ZSHELL to have the LIST command list the file with tabs expanded and pipe the output into the editor.

## INSTALLING ZSHELL

ZSHELL is a module that interfaces with the resident portion of DOS. ZSHELL loads and relocates itself to high memory. It protects itself by lowering the HIGH\$ contents. Thus, before you can use the additional functions present in ZSHELL, you have to "install" it in high memory. This is done at "DOS Ready" by entering the command:

```
ZSHELL [(DRIVE=d,BREAK=sw,LENGTH=n,DISABLE)]
```

where:

|          |                                                                                                         |
|----------|---------------------------------------------------------------------------------------------------------|
| DRIVE=d  | Place piping files on drive d with d being a valid drivespec <0-7>.                                     |
| BREAK=sw | An option to send or omit a BREAK character upon reaching the end of the input file. Default is OFF     |
| LENGTH=n | Establishes the length of the SHELL command line [255 max]. It defaults to the DOS command line length. |
| OFF      | Is used to disengage the resident ZSHELL module and attempt to reclaim the high memory it used.         |

Abbreviations: DRIVE=D, DISABLE=DIS=OFF=N

Note: Parameters within brackets "[]" are optional.

Typing ZSHELL from DOS Ready will cause ZSHELL to load, relocate itself to high memory, and then become active. After ZSHELL has been activated it will remain tied into the system until the computer is rebooted. A "global reset" CANNOT remove ZSHELL from the system. This is because ZSHELL is connected to the resident system module in addition to the devices. RESET cannot undo this "connection".

The parameter, DISABLE, will be discussed first. Since ZSHELL is tied into the system and cannot be removed by RESET, the OFF parameter serves this purpose. Once ZSHELL is activated, if you want to de-activate it, simply enter the command:

ZSHELL (OFF)

ZSHELL will unhook itself from the system. If no other module was placed into high memory after ZSHELL was installed, then ZSHELL will reset HIGH\$ to the value that existed prior to its installation. Thus, the high memory space it used will be freed for subsequent use.

The concept of piping is normally implemented on larger computers by executing the "interconnected" programs simultaneously. This is known as "multiprocessing". With it, a channel of communications is established between the two programs - it is this channel that is termed the "pipe". Your

computer environment does not support "multiprocessing". Therefore, piping has been implemented by chaining one program to another with the "standard output" of the first temporarily stored in a holding file until the first program completes its execution. The second program then uses this holding file as its "standard input". Since the second program could have standard output "piped" to a third program, a second holding file is needed. The parameter, DRIVE, will allow you to instruct ZSHELL on which drive to place these piping files. DRIVE will default to drive=0 if you omit the parameter.

The DRIVE parameter can be changed by simply reinstalling ZSHELL with a new parameter string. Any parameter not entered on the command line will be left unchanged. The new copy of ZSHELL will re-use the same space in memory if it finds itself already installed.

The BREAK parameter is associated with redirection of standard input. Its use will be described in that section.

The LENGTH parameter is used to establish the size of the command line buffer which ZSHELL maintains. This can be up to 255 characters in length. The minimum is the DOS limit (63 for Model I/III, 79 for DOS 6). A longer line buffer permits the entry of complex piping commands as well as more individual commands connected with the ";" multiplier. However, the command line passed to the DOS after parsing by ZSHELL and extraction of redirection segments can not exceed the DOS limit.

## COMMAND LINE SYNTAX

While active, ZSHELL monitors the command line entered in response to the "DOS Ready" prompt. If the first character of the command line is a double-quote, '"', then the command line will be passed unaltered to the command interpreter. This could be useful for those LC users wanting the application to invoke any redirection desired. There are several special characters that, when entered on the command line, will cause ZSHELL to take appropriate action. Let's take a look at each character and see what it does.

### Redirect STANDARD OUTPUT: >, >+, >>, >>+

The first character is the greater-than symbol or right caret, ">". When the ">" symbol is detected on the command line, ZSHELL will cause the \*DO standard output to be redirected to the devicespec or filespec which follows the greater-than symbol. If the caret is followed by the plus sign, "+", then the \*PR device is considered to be the standard output device. For example:

```
DEVICE >*PR
```

will direct the display of the DEVICE command to a printer instead of the video display. Alternatively,

```
DIR :0 (A,I,P) >+CATALOG/TXT
```

will direct the "printer" output of the DIRectory command to a disk file named, CATALOG/TXT. Do NOT attempt to redirect standard output to the \*DO device. When you use the ">+" construct, do NOT redirect to the \*PR device.



## ZSHELL Command Line Processor

If the ">" symbol is followed by another greater-than symbol, the output will be APPENDED to the devicespec or filespec. For example:

```
>>MAP/DAT FREE :1
```

will append the drive 1 free space map to the end of the file, MAP/DAT. Note that this example shows the redirection specification first. Actually, the redirection specification can be anywhere on the command line except in the middle of a parameter string. Thus, the following are equivalent command statements:

```
>>MAP/DAT FREE :1
FREE >>MAP/DAT :1
FREE :1 >>MAP/DAT
```

Isn't that flexibility? Remember that the standard output will be assumed to be the \*DO device. The \*PR device can be easily used as "standard output" by simply adding a plus sign immediately after the right caret.

### Redirect STANDARD INPUT: <, <#, <@

The second character is the less-than symbol or left caret, "<". The less-than symbol causes ZSHELL to redirect standard input. Programs that take input generally have a couple of different methods of noting when no more input is available. Some expect you to depress the <BREAK> key when you have completed your input (e.g. the DOS BUILD library command). Other programs are looking for a specific sequence of characters to signify the end of input. In this case, if an end-of-file is reached, it is an error and the program should abort. There are also programs which do not expect to ever see an "end-of-file" condition from their "standard input" (i.e. BASIC). To satisfy these three classical ways of handling input terminating conditions, ZSHELL provides three forms of input redirection. All three forms cause standard input to be redirected according to your device or file specification but cause different things to occur if an end-of-file is reached.

If the "<" symbol is entered immediately followed by the devicespec or filespec, then the standard input will be retrieved from that file or device. When, and if, the end-of-file is reached, ZSHELL will automatically disengage the redirected input and a <BREAK> character will be sent through the standard input before control is returned to the \*KI. The BREAK bit in the KFLAG\$ will also be set (the KFLAG\$ is documented in the technical section of your DOS manual). If you do not program in assembly language, you need not bother with the KFLAG\$. This BREAK will allow a program to detect the end-of-file (EOF) condition by scanning for a break.

If the "<" symbol is followed by a number sign or pound sign, "#", an end-of-file is handled somewhat differently. The control of the standard input will be returned to the \*KI device as noted above; however, the original \*KI handling will be restored. Thus, the standard input is retrieved once again from the original \*KI device. The pound sign character, "#", should be easy to remember if you correlate "pounding" on something as "breaking" it.

Passing an EOF is the "UNIX" way of handling input redirection. If you find it convenient for use in your programs to have standard input revert to

the normal keyboard device without having to use the pound sign, you can request this at the time you install ZSHELL by specifying the option:

ZSHELL (BREAK=OFF)

with BREAK=OFF installed, ZSHELL will invert the sense of the pound sign appendage - "<" passes control to \*KI while "<#" adds the BREAK character.

The last form of standard input redirection is the "<" symbol followed by an "@" symbol. If you had specified a command entry of the form:

BASIC RUN "BATCH" <@DATAFILE/TXT:2

and an end-of-file was reached, ZSHELL will abort the current program by transferring control to the @ABORT vector. If Job Control Language (JCL) was active, the JCL will abort. Use this form of redirection if your program should not expect to read to the end of a file and would, in fact, be an error if it did. Remember the "@" to indicate @ABORT since it is the first character of "@ABORT".

Pipe OUTPUT (1) to INPUT (2): |, |+, |#, |@, |#+, |@+

The third character is the vertical brace, "|". The vertical brace may be generated by simultaneously depressing <CLEAR><SHIFT></>. The "|" will cause ZSHELL to "pipe" the display output of the first command (\*DO output) to the input of the second command. There may be any number of commands that are separated by the vertical brace which continue to pipe from left to right. You are limited by the maximum size of the command line [this is one reason for a larger command line length at ZSHELL installation time]. For example, a command line of:

DIR :0 (A,I,S,N) | LSCRIPT

will pipe the output of the directory display into LSCRIPT for immediate word processing. It is important to observe the parameter, "N", and why it is specified. Remember that certain programs presenting data on the display screen may pause when the screen is filled. These programs are expecting some keyboard response (typically an <ENTER>) before continuing. Just because you have redirected the display output to a printer or disk file, it does not eliminate the need for such a response. Programs that present output pauses should have options (such as the "N" parameter in the DIR command) to allow non-stop output. If you observe that a particular program may have stopped its output, it may be because of this pause. You may want to "nudge" it along by depressing <ENTER> or some other response as dictated by the program you are running. Without ZSHELL, to achieve the identical results as the above example, you would have had to enter the sequence:

```
ROUTE *PR TEMP/TXT:0
DIR :0 (A,I,S,P)
RESET *PR
LSCRIPT
<SHIFT><ENTER> L TEMP/TXT
```

to accomplish the same function more easily provided by ZSHELL.



## ZSHELL Command Line Processor

The \*PR device output can be used for piping standard output instead of the \*DO device by immediately following the vertical bar with a plus sign, "+". This is done just as easily as it was done for output redirection. You may also use the pound sign and abort options with pipe commands just like they were illustrated for the redirection of STANDARD INPUT. The options are used to effect the result of an end-of-file detected on the device stream by the program accepting input from the pipe.

### Multiple commands: ;

The last character is the semi-colon, ";". ZSHELL will allow the entering of more than one command on a single command line. Just separate the commands with a semi-colon and ZSHELL will handle the rest. For example,

```
LIST TEST1/ASM ; LIST TEST2/ASM; LIST TEST3/ASM:1
```

In this example, note that spaces may be inserted either before or after the semicolon. ZSHELL will disregard any spaces surrounding the semicolon as well as the vertical bar (piping). Again be aware that the redirection specifications can occur anywhere on the command line except within any parameter string. ZSHELL ignores all characters found between a left parenthesis, "(", and its closing right parenthesis, ")". Parentheses are used to surround the parameter strings. Note that most versions of DOS permit you to omit the closing right parenthesis. However, if you are going to follow your parameters with redirection specifications, you must close your parameters with the right parenthesis so ZSHELL can accept your specifications. For example:

```
DIR (A,I,S,N >CATALOG/TXT:3
```

will not redirect the output of the DIRectory command since the parameters are not "closed". The command should be entered in one of the following forms:

```
DIR >CATALOG/TXT:3 (A,I,S,N  
>CATALOG/TXT:3 DIR (A,I,S,N  
DIR (A,I,S,N) >CATALOG/TXT:3
```

### ADVANCED TOPICS

Although you cannot redirect the standard input or output of a DO command (ZSHELL restriction), you may use redirection from within a Job Control Language (JCL) file. If standard input is redirected in a "command line" from within a JCL file, then ZSHELL will cause standard input to be retrieved from the specified file or device and NOT from the JCL file. This retrieval will continue until the particular application execution has terminated or until end-of-file is reached. If the standard input redirection is accomplished using the "<" specification, JCL may or may not ABORT on reaching end-of-file depending on where within the execution the <BREAK> generated by ZSHELL was detected. If the standard input redirection is accomplished using the "<#" specification, then the JCL will continue where it left off upon reaching end-of-file. Note that the two preceeding results will be reversed if "BREAK=ON" is installed. If the "<@" option is used, then the JCL will be terminated when the standard input reaches end-of-file.



A few words are necessary concerning the use of the pound sign, "#". JCL already uses the "#" to indicate a substitution field when using compiled JCL. Therefore, if you are going to compile your JCL file, you must enter two "#"s in a row to let JCL know you indeed meant to enter a "#". This means that you specify "<##INPUT/TXT". If, on the other hand, you are going to execute your JCL file without compilation (i.e. DO = ), then use only a single "#".

It is very simple to write programs that utilize the abilities of ZSHELL. All input should be retrieved from standard input (\*KI) and all output should be sent to the standard output. In BASIC, this would be "INPUT" and "PRINT" statements. In assembly language, you would use @KEY, @KEYIN, @DSP, and @DSPLY calls. The program may then be used with ZSHELL to allow you to redirect the input and the output. One version of a program may then access any device or file for input or output without any modifications.

For those of you who own LC, the C language compiler available from MISOSYS, specify the #OPTION REDIRECT OFF in your LC source program. Since ZSHELL will handle all command line I/O redirection, the code output by the compiler to handle redirection would never be used and only would serve to lengthen the object program unnecessarily.

When using the redirection of standard input, one should be aware that this feature may not always work as expected because of certain programming techniques. Take for an example a program that checks for a <BREAK> by constantly invoking @KBD and checking for the BREAK character. The program will be retrieving keystrokes which may have been meant for later responses. To deal with this problem, your programs should check for the <BREAK> key by checking the status of the BREAK bit contained in the KFLAG\$. Consult your DOS manuals for programming details.

To perform the piping functions, ZSHELL uses two files called ZO/PIP and ZI/PIP. These files are created by ZSHELL when you have entered PIPING specifications and are used during the execution of the programs being piped. They will no longer be used by ZSHELL once the piping is completed and may be deleted later if you don't wish to have them cluttering up your disk. Do NOT delete either one of these files while piping is active.

Under Model I/III operation, ZSHELL acts like a filter on the keyboard. It also intercepts the @KEYIN call originally executed by SYS1. Because of this, if you are using the LDOS MiniDOS filter and ZSHELL is installed, the "repeat-last-DOS-command" function, <CLEAR><SHIFT><R>, will be inoperative.

## ERROR MESSAGES EXPLAINED

Approximately half of the resident portion of ZSHELL is taken up by file buffers and control blocks. Input and output each take a 256-byte buffer and a 32-byte File Control Block. Due to these memory requirements, the error messages have been kept short to avoid wasteful use of high memory. Because of this, there are only two error messages. They are explained below along with examples.

## Redirection error

There are four major causes for this error message.

1. An attempt was made to redirect standard input or standard output more than once on a command. ZSHELL could redirect the standard output to a file or a device but NOT to both at the SAME time.

BASIC >\*PR >TEST/TXT

2. An attempt was made to redirect standard input or standard output on a DO command. Redirection is possible within a JCL but NOT on the command that initiates that JCL.

DO JOB/JCL <INPUT/TXT

3. An attempt was made to redirect standard input to be retrieved from the \*KI device. Standard input normally comes from the \*KI.

LSCRIPT <\*KI

4. An attempt was made to redirect standard output to the standard output device. When ">" is used, you cannot redirect the standard output to the \*DO and when ">+" is used, you cannot redirect the standard output to the \*PR.

LSCRIPT >\*DO

## Piping error

There are three major causes of this error message.

1. An attempt was made to redirect standard input while the standard input was already being piped from a previous command.

DIR :0 | LSCRIPT <INPUT/TXT

2. An attempt was made to redirect standard output while the standard output was being piped to a following command.

DIR :0 >OUTPUT/TXT | LSCRIPT

3. An attempt was made to invoke @CMNDI while piping was active. If you want to execute a "/CMD" program from within an executing program, then use the @RUN system call instead of @CMNDI. The @CMNDI call is needed by ZSHELL to supervise the redirection function.

## Parameter error

This is not an error message displayed by ZSHELL. It is possible to get this error from your application if you inadvertently forget to close your parameter string entries with the right parenthesis when you follow the parameters with redirection specifications.

## KISTORE/FLT

KISTORE is a filter that will allow the simultaneous copying of all \*KI keystrokes to a file or device. This filter will be useful in conjunction with ZSHELL's ability to retrieve standard input from a disk file. In order to install the filter, it is necessary to enter the command:

```
FILTER *KI to KISTORE [(REWIND)] (Model I/III)
```

```
SET *KS to KISTORE [(REWIND)] | (DOS 6.x)
FILTER *KI *KS |
```

REWIND            Is used to force the storage file to be rewound to its beginning so that any contents of an existing file are ignored.

Abbreviations: REWIND=R

Note: Parameters within brackets "[]" are optional.

Once KISTORE/FLT is established, the filter remains dormant until activated. Simultaneously depressing <CLEAR><SHIFT><O> will activate the filter. You must be using the DOS keyboard driver to activate and deactivate the filter. KISTORE will save the contents of the last two lines of the video and display the prompt:

## Filespec?

Enter the filespec that you want to use for the storage of the keystrokes. KISTORE will restore the contents of the last two video lines and begin the copying of keystrokes to the specified file. All keystrokes are appended to any already in the file (assuming the file was existing). This permits you to separate the storage into more than one session - with each session concatenating additional keystrokes to those already stored. If you want to reuse an existing file and ignore its contents, then specify the REWIND parameter when you install KISTORE.

The copying may be terminated by simultaneously depressing <CLEAR><SHIFT><X> and the file will be closed.

To use KISTORE to create a standard input file, run the application, depress <CLEAR><SHIFT><O> to open the storage file. Proceed to use the application as you would normally. When you are finished, exit to "DOS Ready" if applicable, and depress <CLEAR><SHIFT><X> to terminate the storage and close the file. The same application may then be executed later using the file as standard input and your computer will be able to duplicate all of your commands without you having to retype them.



## WILDCARD

WILDCARD is a "shell" processor that allows you to invoke compatible commands on a number of file specifications that match a wildcardspec entered on the command line. You enter the command line once while the WILDCARD shell processor searches the designated disk drive(s) for files that match your wildcard specification. WILDCARD builds a Job Control Language file of your command line (minus the "WC") substituting each matching file specification for the wildcard specification on a separate command line. WILDCARD then automatically invokes the JCL file. You can invoke a WILDCARD command with:

|                                       |                                                                                                         |
|---------------------------------------|---------------------------------------------------------------------------------------------------------|
| WC COMMAND wildcardspec command-parms |                                                                                                         |
| COMMAND                               | Is any DOS command using the syntax "COMMAND filespec ...".                                             |
| wildcardspec                          | Is the wildcard specification used to match on-line disk files. The wildcard syntax is described below. |
| command-parms                         | Is any additional entries needed for the "COMMAND".                                                     |

The "wildcardspec" uses the file name and file extension as two distinct fields for matching purposes. If the drive specification is entered, WC will search that specific drive for all files matching the name-extension wildcard fields. If the drive specification is omitted, then all drives will be searched. Within each field, WC accepts two wild characters, "?" and "\*". The question mark will match any character in that character position. The asterisk is used to match all trailing characters in the field. For example, "?SHELL/TXT:1" will match with ASHELL/TXT, BSHELL/TXT, etc. but ASHELL1/TXT will not match. A global match of all filespecs would be an entry of the form, "\*/\*"; whereas a match of all /CMD files would be an entry of the form, "\*/CMD". If a minus sign, "-", precedes the filename field, WILDCARD will select files that do NOT match the wildcardspec. Any entered password will be used in the full file specifications generated by the selection process. Note that this wildcard syntax is different from the DOS partspec!

WC is quite useful to perform repetitive tasks on files whose file specifications are similarly constructed. For example, to list out all /TXT files on drive 1, you could use a WILDCARD entry of:

WC LIST \*/TXT:1

What DOS commands are compatible? All of the following DOS commands are: APPEND, ATTRIB, LIST, LOAD, REMOVE/KILL, RENAME, RESET(V6), and RUN. Other programs that expect a filespec on the command line are also "compatible".

A word of caution - WC uses the SYSTEM/JCL Job Control Language file; therefore, WC cannot be invoked from JCL!

## LSQFB

This utility is designed to allow for a backup with format to be performed. Only floppy drives may be used, and the backup performed must be mirror image. The syntax is:

```
=====
LSQFB :s :d (parm,parm,parm)

:s   is the Source drive. The colon is optional.
:d   is the Destination drive. The colon is optional.

The following optional parameters may be used:

ALL=  parameter used to specify whether all cylinders
      of the source disk will be read and copied to
      the destination disk, or only allocated
      cylinders will be used. The switch ON or OFF
      may be specified, with the default being OFF.

V1=   parameter used to specify whether or not a
      verify of the destination disk is to be
      performed on the 1st pass. The switch ON or OFF
      may be used, with the default being ON.

V2=   parameter used to specify whether or not a
      verify of the destination disk is to be
      performed on the 2nd pass. The switch ON or OFF
      may be used, with the default being OFF.

QUERY= Query for parameters not specified. The switch
        ON or OFF may be used. The default is OFF

abbr:  ON=Y, OFF=N, QUERY=Q, ALL=A
=====
```

The LSQFB (Quick Format and Backup) utility will allow for the creation of a mirror image backup of a source disk without having to format the destination disk prior to executing the backup. The normal means by which a mirror image backup is made using LDOS/TRSDOS 6.x is to first format a diskette using the FORMAT utility, and then use the BACKUP utility to perform the backup. The limitations of LSQFB are as follows:

- 1.) Two distinct FLOPPY drives must be used.
- 2.) The source diskette must have been formatted using the LDOS/TRSDOS 6.x FORMAT utility, and cannot contain any non-standard format.

LSQFB will perform a "single pass" format and backup. If LSQFB is entered with no drives specified, prompts will appear for them. If drive numbers are specified, the first drive number will represent the source drive, and the destination drive will be the second drive number. If no parameters are specified, the defaults will be used.

Consider the results of entering the following command.

LSQFB 1 2

Drive 1 will be used as the source drive, while drive 2 will be the destination drive. Prior to LSQFB performing any action, a prompt will appear to load the diskettes. Once the proper diskettes have been installed, press <ENTER>, and the backup will begin. The following actions will take place.

- 1.) The source diskette will be logged in, to determine the type of format.
- 2.) Cylinder 0 of the destination diskette will be formatted.
- 3.) If cylinder 0 of the source disk contains data, it will be read into memory.
- 4.) If cylinder 0 of the source diskette contains data, the information stored in memory (see Step 3) will be written out to the destination diskette.
- 5.) Cylinder 0 of the destination diskette will be verified.
- 6.) Steps 2-5 will be repeated for all remaining cylinders.
- 7.) The following message will appear after the last cylinder has been verified:

Duplication complete      1 disk    created

Replace destination disks and press <ENTER> to repeat  
..<R> to restart with new parameters  
...or....<BREAK> to exit program.

- 8.) Press <ENTER> in response to this prompt to make another mirror image backup.  
Press <BREAK> to abort the LSQFB utility. The following prompt will appear:

Load SYSTEM diskette and hit <ENTER>

Place a system diskette in drive 0 and press <ENTER>, to return to the DOS level.

If it is desired to use LSQFB again with different parameters, press <R> in response to the prompt displayed in step 7. Doing so will cause the drives to be prompted for, and prompts will appear for all parameters.

If LSQFB is to be restarted, or the command LSQFB (Q=Y) is entered, the following prompts for the parameters will occur:

Duplicate unallocated tracks? (Y/N)  
Verify on same pass? (Y/N)  
Verify on second pass? (Y/N)

The first prompt relates to the ALL parameter. If it is answered with <Y>, all cylinders will be read from the source diskette and written to the destination diskette, regardless of whether or not the cylinder contains information. If this prompt is answered <N>, only cylinders containing information will be read and written.

The next prompt relates to the V1 parameter. If it is answered with <Y>, all cylinders on the destination diskette will be verified immediately after all writes. If answered <N>, no immediate verify will be done.

The final prompt corresponds to the V2 parameter. If it is answered with <Y>, all cylinders on the destination diskette will be verified upon completion of all writing to the diskette. If answered <N>, there will be no second pass verification.

If an error occurs, an appropriate error message will be displayed, and a prompt will appear requesting the course of action that is desired. During any LSQFB operation, the <BREAK> key will be active, and can be used to abort the process.

#### I M P O R T A N T

LSQFB assumes that a mirror image backup is desired, and performs no check on the destination diskette with respect to the existence of data. Any existing information on a destination diskette will ALWAYS be destroyed. Also, LSQFB will NOT clear the Mod Flags of files on the source diskette.



## L S C O M P

Compares two files, parts of files, diskettes, or parts of diskettes for a character for character match. The proper syntax is :

```
=====
| LSCOMP filespec1 TO filespec2 (parm,parm,...)
| LSCOMP :drive1 TO :drive2 (parm,parm,...)
|
| The allowable parameters are :
|
| REC=      Starting record number of the filespecs at
|            which the compare will begin (default is 0).
|
| NUM=      Number of records of a filespec or sectors of
|            a disk to compare.
|
| ALL       Display each non-matching byte.
|
| PRINT     Send display to *PR as well as *DO.
|
| CYL=      Cylinder at which to start compare between two
|            drives (default is 0).
|
| SEC=      Starting sector of a diskette to compare
|            (default is 0).
|
| Abbr: REC=R, NUM=N, ALL=A, PRINT=P, CYL=C, SEC=S
|
=====
```

This utility compares two files or two entire diskettes to determine whether or not the information in or on them is identical. It is usually performed after a BACKUP or a COPY to determine the validity of the data.

Observe the following command and output that is generated:

```
LSCOMP MAY/DAT:3 :4
```

```
MAY/DAT:3    contains    17 sectors, EOF offset = 70
MAY/DAT:4    contains    17 sectors, EOF offset = 70
```

Notice that the second filespec was indicated by a drivespec. This is the ONLY exception to a complete filespec which is allowed. Since the files proved to be identical, only the number of compared sectors followed by the end-of-file offset were displayed.

In the case of differing files the following would occur :

```
LSCOMP FISCAL82/DAT:3 FISCAL82/DAT:4 (R=4)
```

```
Posn= X'0005,00  FISCAL82/DAT:3    = X'20,  FISCAL82/DAT:4    = X'00
                29 bytes did not match.
```

```
Posn= X'0005,B0  FISCAL82/DAT:3    = X'54,  FISCAL82/DAT:4    = X'00
                32 bytes did not match.
```

```
FISCAL82/DAT:3    contains    18 sectors, EOF offset = 100
FISCAL82/DAT:4    contains    18 sectors, EOF offset = 100
```

The display shows the record number of a discrepant sector followed by the relative byte, and the contents of that byte in each filespec. The second line displays the total number of subsequent bytes which do not match. If the ALL parameter had been specified, each of the sixty-one bytes would have been displayed in the first format. Since the parameter R=4 was specified, the compare began at record 4.

To compare one disk to another use drive numbers instead of filespecs. The starting cylinder and sector number may be specified either in X'00' format or as a decimal integer. The number of contiguous sectors to compare may also be specified by using the NUM= parameter.

Unlike file to file comparisons, the disk to disk compare will display the currently accessed sector relative to the current cylinder. As much information as possible will be read into memory from the source drive (the first drivespec). This information will then be compared to the destination drive. If discrepant bytes are detected the following will appear on the video:

Cyl X'0D, Sec X'00, Byte X'00, Drive 2 = X'6D, Drive 3 = X'31  
3078 bytes did not match.

If the ALL parameter had been specified then each different byte would display in the first line format. To send the output to the printer as well as the video, specify the PRINT parameter.